Quantitative Monitor Automata

Krishnendu Chatterjee[1(✉)], Thomas A. Henzinger[1], and Jan Otop[2]

[1] IST Austria, Klosterneuburg, Austria
krish.chat@gmail.com
[2] University of Wroclaw, Wroclaw, Poland

Abstract. In this paper we review various automata-theoretic formalisms for expressing quantitative properties. We start with finite-state Boolean automata that express the traditional regular properties. We then consider weighted ω-automata that can measure the average density of events, which finite-state Boolean automata cannot. However, even weighted ω-automata cannot express basic performance properties like average response time. We finally consider two formalisms of weighted ω-automata with monitors, where the monitors are either (a) counters or (b) weighted automata themselves. We present a translation result to establish that these two formalisms are equivalent. Weighted ω-automata with monitors generalize weighted ω-automata, and can express average response time property. They present a natural, robust, and expressive framework for quantitative specifications, with important decidable properties.

1 Introduction

In this work we review various automata-theoretic formalisms for expressing quantitative properties. We start with the motivation for quantitative properties.[1]

Traditional to quantitative verification. The traditional formal verification problem considers Boolean or functional properties of systems, such as "every request is eventually granted". For analysis of resource-constrained systems, such as embedded systems, or for performance analysis, quantitative properties are necessary. Hence recently significant research activities have been devoted to quantitative aspects such as expressing properties like "the long-run average success

This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-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.

[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.

X. Rival (Ed.): SAS 2016, LNCS 9837, pp. 23–38, 2016.
DOI: 10.1007/978-3-662-53413-7_2

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".

Automata-based properties. Automata have been one of the standard ways to express specifications of system properties. For example, for Boolean properties, automata provide a robust way to express all ω-regular properties [28], and all formulas expressed in Linear-time Temporal Logic (LTL) can be translated to finite-state ω-automata [26]. We review in this paper various automata-based frameworks to express quantitative properties.

Natural ways for extension. The first natural way to express quantitative properties is to consider automata with counters. However, computational analysis of such models quickly leads to undecidability (such as two-counter machines), and a classical way to limit expressiveness for decidability is to consider *monitor counters*, i.e., the counter values do not influence the control. The second approach is to consider automata with weights (or weighted automata). We describe below various approaches that explore the two above possibilities.

Weighted automata over finite words. The first extension of automata with weights was considered as weighted automata over finite words, where the weights come from a semiring [22]. The weighted automata framework for example can express the worst-case execution time, where every transition is labeled with a weight that represents the instruction execution time, and the automaton can choose the supremum over all traces. The weighted automata framework has been significantly enhanced as cost register automata [2]. However, both the weighted automata and the cost register automata framework are restricted to finite words only. In this work we focus on automata over infinite words.

Weighted ω-automata. The counterpart of weighted automata for infinite words is called *weighted ω-automata* (also referred to as *quantitative automata* in [14]). In weighted ω-automata, which extend finite automata, every transition is assigned a rational number called a weight. Hence every run gives rise to an infinite sequence of weights, which is aggregated into a single value by a value function. For non-deterministic weighted ω-automata, the value of a word w is the infimum value of all runs over w. Weighted ω-automata provide a natural and flexible framework for expressing quantitative properties [14]. For example, the property of long-run average of the ratio of requests to grants can be expressed with weighted ω-automata, but not weighted automata over finite words. However, even weighted ω-automata cannot express the following basic property [18].

Example 1. Consider infinite words over $\{\boldsymbol{r}eq, \boldsymbol{g}ra, \#\}$, where $\boldsymbol{r}eq$ represents requests, $\boldsymbol{g}ra$ represents grants, and # represents idle. A basic and interesting property is the average number of #'s between a request and the corresponding grant, which represents the long-run average response time of the system. We consider two variants: first, when at any point at most one request can be pending, and in general, arbitrarily many requests can be pending.

Weighted automata with monitors. To express quantitative properties such as average response time, weighted ω-automata can be extended with monitors.

The monitors can be of two types: (a) counters; and (b) weighted automata over finite words. We explain the two approaches below.

Automata with monitor counters. Automata with monitor counters are similar in spirit to cost register automata, but for infinite words. They are automata equipped with counters. At each transition, a counter can be started, terminated, or the value of the counter can be increased or decreased. However, the transitions do not depend on the counter values, and hence they are referred to as monitor counters. The values of the counters when they are terminated give rise to the sequence of weights. A value function aggregates the infinite sequence into a single value. Automata with one monitor counter can express the average response time property with at most one request pending (in general at most k requests pending with k counters), which weighted ω-automata cannot.

Nested weighted automata. The second way to enrich expressiveness of weighted ω-automata is to consider weighted automata over finite words as monitors. This gives rise to the framework of *nested weighted automata (NWA)* [18]. An NWA 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. In other words, the slave automata are weighted automata over finite words, whereas the master automaton is an weighted ω-automaton. For Boolean finite automata, nested automata are equivalent to the non-nested counterpart, whereas NWA are strictly more expressive than non-nested weighted ω-automata [18], for example, NWA can express the long-run average response time property (see [18, Example 5]). The NWA framework provides a specification framework where many basic quantitative properties, that cannot be expressed by weighted ω-automata, can be expressed easily, and it provides a natural framework to study quantitative run-time verification (see [18, Sect. 5]). NWA can express the average response time property with any number of requests pending.

The relationship. We establish a close relationship between NWA and automata with monitor counters. More precisely, we show that automata with monitor counters form a special class of NWA. An NWA has width k if at any point at most k slave automata can be active. An NWA with bounded width is an automaton that has width at most k, for some k. We show that the class of automata with monitor counters exactly coincides with the class of NWA with bounded width. Thus the two different ways of adding monitors to weighted ω-automata coincide and give a robust formalism for quantitative specifications. Note that NWA in general allow to have unbounded number of monitor counters, and are more interesting as a theoretical framework, and also more challenging to establish decidability results.

Decidability results. Finally, several interesting computational problems for NWA are decidable [18]. For example, for a subclass of NWA that can express the average response time property, the fundamental automata theoretic questions of emptiness and universality are decidable [18, Theorem 15].

Summary. In summary, NWA provide a natural, expressive, and robust specification framework for quantitative properties: it is a natural extension of weighted ω-automata with nesting, which can express basic performance properties such as average response time, and robust since two different formalisms of monitors lead to the same expressive power. Moreover, it enjoys nice computational aspects, such as decidability of the basic questions. A pictorial description of the landscape of the various automata theoretic formalisms to express quantitative properties is shown in Fig. 1.

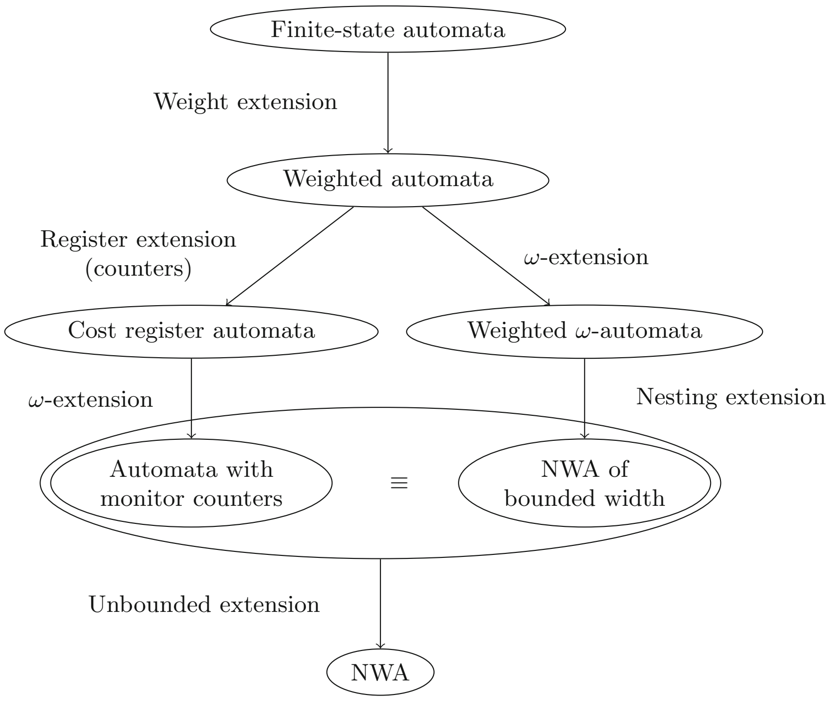

Fig. 1. An overview of the automata models discussed in this paper. The arrows are labeled with a new feature introduced by a more general model, i.e., assigning values to words (weight extension), allowing infinite words (ω-extension), allowing counter monitors (register extension), allowing nesting (nesting extension), and allowing unbounded number of monitors (unbounded extension).

Related works. While in this work we focus on weighted automata for infinite words, we briefly mention the other related works. Weighted automata over finite words (see the book [22] for an excellent collection of results) as well as infinite words without monitors [13,14,23] have been extensively studied. The extension to weighted automata with monitor counters over finite words has been considered as cost register automata in [2]. A version of NWA over finite words has been studied in [6]. NWA over infinite words were introduced in [18]. NWA with bounded width have been studied in [16]. Several quantitative logics have also been studied, such as [1,5,7]. While we will consider the basic decision problems of emptiness and universality for weighted ω-automata, the study of quantitative properties on other models (such as probabilistic models) or other semantics (like probabilistic semantics) have also been studied extensively [3,4,8–11,15,17,19,20,24,25,27].

2 Finite State Automata

In this section we consider Boolean finite-state automata to express qualitative (i.e., functional or regular) properties.

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 $w[i]$. The length of a finite word w is denoted by $|w|$; and the length of an infinite word w is $|w| = \infty$.

Automata. An *automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, Q_0, \delta, F\rangle$, 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, and (5) F is a set of accepting states, An automaton $\langle \Sigma, Q, q_0, \delta, F\rangle$ is *deterministic* if and only if δ is a function from $Q \times \Sigma$ into Q and Q_0 is a singleton. In definitions of deterministic automata we omit curly brackets in the description of Q_0 and write $\langle \Sigma, Q, q_0, \delta, F\rangle$.

Semantics of automata. A *run* π of an automaton $\mathcal{A}$ on a word w is a sequence of states of $\mathcal{A}$ of length $|w|+1$ such that $\pi[0]$ belong to the initial states of $\mathcal{A}$ and for every $0 \leq i \leq |w|-1$ we have $(\pi[i], w[i], \pi[i+1])$ is a transition of $\mathcal{A}$. A run π on a finite word w is *accepting* iff 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* iff 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. Finally, we define a language *recognized* by an automaton $\mathcal{A}$ as the set of words w such that $\mathsf{Acc}(w) \neq \emptyset$.

Example 2 (Responsiveness). We consider a system with three types of events: requests ($\boldsymbol{r}eq$), grants ($\boldsymbol{g}ra$) and null instructions (#). We consider the following specifications express that the system is responsive:

SB1 $\mathbf{G}(\boldsymbol{r}eq \rightarrow \neg \boldsymbol{r}eq \mathbf{U} \boldsymbol{g}ra)$: *requests and grants are organized in non-overlapping pairs*

SB2 $\mathbf{G}(\boldsymbol{r}eq \rightarrow \mathbf{F}\boldsymbol{g}ra)$: *every request is followed by a grant,*

SB3 $\mathbf{GF}\boldsymbol{r}eq \rightarrow \mathbf{GF}\boldsymbol{g}ra$: *if infinite number of request is issued, the system issues infinitely many grants*

The above properties are ordered from the strongest to the weakest, i.e., the first property implies the second and the second implies the third. The converse implications do not hold.

Every LTL formula φ can be translated to a Büchi automaton, which recognize words satisfying φ. In particular, properties SB1, SB2 and SB3 can be expressed by automata presented in Fig. 2.

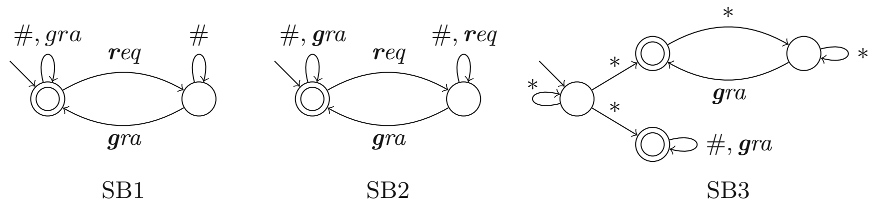

Fig. 2. Finite-state automata expressing properties SB1, SB2 and SB3. Double circles denote accepting states, the state the most to the left is an initial state and $*$ denotes all letters $\boldsymbol{r}eq, \boldsymbol{g}ra, \#$.

Results. A basic question for automata is the language emptiness (resp., universality) problem that asks whether there exists a word that is accepted (resp., all words are accepted). The emptiness problem is NLOGSPACE-complete whereas the universality problem is PSPACE-complete [28].

3 Weighted Automata

In this section we present weighted automata over finite words and infinite words. We call weighted automata over infinite words as weighted ω-automata (which was originally called *quantitative automata* [14]). For brevity, if it is clear from the context that we consider infinite words, then we simply use weighted automata instead of weighted ω-automata. These automata assign numbers to words, and hence can express quantitative properties such as *workload* of a system. The following definitions are common for finite- and infinite-word automata.

Labeled automata. For a set X, an *X-labeled automaton* is an automaton extended by a function C assigning elements of X to transitions of $\mathcal{A}$. Formally, X-labeled automaton $\mathcal{A}$ is a tuple $\langle \Sigma, Q, Q_0, \delta, F, C\rangle$, where $\langle \Sigma, Q, Q_0, \delta, F\rangle$ is an automaton and $C : \delta \mapsto X$ is a labeling function.

Weighted automata. A *weighted automaton* is a $\mathbb{Z}$-labeled automaton, where $\mathbb{Z}$ is the set of integers. The labels are called *weights*.

Semantics of weighted automata. 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., $C(\pi) = (C(\pi[i-1], w[i], \pi[i]))_{1 \le i \le |w|}$; 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 an empty set is infinite, i.e., the value of a word that has no accepting run 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.

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:* $\textsc{Max}(\pi) = \max_{i=1}^{n}(C(\pi))[i]$ and $\textsc{Min}(\pi) = \min_{i=1}^{n}(C(\pi))[i]$.
2. *Sum and absolute sum:* the sum function $\textsc{Sum}(\pi) = \sum_{i=1}^{n}(C(\pi))[i]$ and 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.

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

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

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

Example 3 (Workload). Recall the setting of requests and grants from Example 2 and properties regarding responsiveness of the system. However, property 1 is satisfied by a trace $\boldsymbol{r}eq\boldsymbol{g}ra\boldsymbol{r}eq^2\boldsymbol{g}ra \ldots \boldsymbol{r}eq^i\boldsymbol{g}ra \ldots$ in which the average number of requests per grant tends to infinity. Property 1 implies the long-time average of requests per grant is 1, but it is much stronger as it requires a pending request to be matched with a grant before a new request can be issued. With $\textsc{LimAvg}$-automata we can specify the *workload* of the system, which is defined as the long-term average of difference between requests and grants.

Results. The classical decision questions for automata, emptiness and universality have their counterparts in the quantitative framework. The emptiness (resp., universality) problem that asks, given a weighted automaton and a threshold, whether there exists a word whose value does not exceed the threshold (resp., values of all words do not exceed the threshold). The complexity of these problems depends on the value function. For the finite words, a comprehensive account of the results is available in [22]. Below we discuss the results for infinite words. For the considered values functions, the emptiness problem is in PTIME [14, Theorem 3], whereas complexity of the universality problem ranges between PSPACE-complete [14, Theorem 7] and undecidable ([12, Theorem 5] which follows from [21, Theorem 4]).

4 Automata with Monitor Counters

In this section we consider automata with monitor counters, which extend weighted ω-automata.

Automata with monitor counters. An *automaton with n monitor counters* $\mathcal{A}^{\text{m-c}}$ is a tuple $\langle \Sigma, Q, Q_0, \delta, F\rangle$ where (1) Σ is the alphabet, (2) Q is a finite set of states, (3) $Q_0 \subseteq Q_0$ is the set of initial states, (4) δ is a finite subset of $Q \times \Sigma \times Q \times (\mathbb{Z} \cup \{s,t\})^n$ called a transition relation, (each component refers to one monitor counter, where letters s, t refer to starting and terminating the counter, respectively, and the value from $\mathbb{Z}$ is the value that is added to the counter), and (5) F is the set of accepting states. Moreover, we assume that for every $(q, a, q', \boldsymbol{u}) \in \delta$, at most one component in $\boldsymbol{u}$ contains s, i.e., at most one counter is activated at each position. Intuitively, the automaton $\mathcal{A}^{\text{m-c}}$ is equipped with n counters. The transitions of $\mathcal{A}^{\text{m-c}}$ do not depend on the values of counters (hence, we call them monitor counters); and every transition is of the form $(q, a, q', \boldsymbol{v})$, which means that if $\mathcal{A}^{\text{m-c}}$ is in the state q and the current letter is a, then it can move to the state q' and update counters according to v. Each counter is initially inactive. It is activated by the instruction s, and it changes its value at every step by adding the value between $-N$ and N until termination t. The value of the counter at the time it is terminated is then assigned to the position where it has been activated. An automaton with monitor counters $\mathcal{A}^{\text{m-c}}$ is *deterministic* if and only if Q_0 is a singleton and δ is a function from $Q \times \Sigma$ into $Q \times (\mathbb{Z} \cup \{s,t\})^n$.

Semantics of automata with monitor counters. A sequence π of elements from $Q \times (\mathbb{Z} \times \{\bot\})^n$ is a *run* of $\mathcal{A}^{\text{m-c}}$ on a word w if (1) $\pi[0] = \langle q_0, \bot\rangle$ and $q_0 \in Q_0$ and (2) for every $i > 0$, if $\pi[i-1] = \langle q, \boldsymbol{u}\rangle$ and $\pi[i] = \langle q', \boldsymbol{u'}\rangle$ then $\mathcal{A}^{\text{m-c}}$ has a transition $(q, w[i], q', \boldsymbol{v})$ and for every $j \in [1, n]$ we have (a) if $v[j] = s$, then $u[j] = \bot$ and $u'[j] = 0$, (b) if $v[j] = t$, then $u[j] \in \mathbb{Z}$ and $u'[j] = \bot$, and (c) if $v[j] \in \mathbb{Z}$, then $u'[j] = u[j] + v[j]$. A run π is *accepting* if some state from F occurs infinitely often on the first component of π, infinitely often some counter is activated and every activated counter is finally terminated. An accepting run π defines a sequence π^W of integers and $\bot$ as follows: let the counter started at position i be j, and let the value of the counter j terminated at the earliest position after i be x_j, then $\pi^W[i]$ is x_j. The semantics of automata with monitor counters is given, similarly to weighted ω-automata, by applying the value function to π^W.

More general counter operations. In our framework, the counter operations we allow are: activation, adding an integer, and termination. More generally, we can allow other operations such as terminating a counter and discarding its value, or adding the value of one counter to another and terminating it. These operations are similar in spirit to the operations on registers with the copyless restriction in the cost register automata [2]. In [2] it has been shown that the copyless sum of registers can be eliminated using a variant of the subset construction. We can apply this construction to our framework, and this more general operations do not add expressive power as compared to the basic operations that we consider. Hence for simplicity, and ease of establishing equivalence with other models, we consider the basic and fundamental set of counter operations for automata with monitor counters.

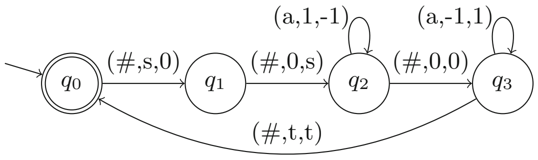

Fig. 3. The automaton $\mathcal{A}_{\text{diff}}$ computing the maximal difference between the lengths of blocks of a's at odd and the following even positions.

Example 4 (Blocks difference [16]*).* Consider an alphabet $\Sigma = \{a, \#\}$ and a language $\mathcal{L}$ of words $(\#^2a^*\#a^*\#)^\omega$. On the words from $\mathcal{L}$ we consider a quantitative property "the maximal block-length difference between odd and even positions", i.e., the value of word $\#^2a^{m[1]}\#a^{m[2]}\#^3\dots$ is $\sup_{0\leq i}|m[2*i+1]-m[2*i+2]|$. This property can be expressed by a Sup-automaton $\mathcal{A}_{\text{diff}}$ with two monitor counters depicted in Fig. 3.

The automaton $\mathcal{A}_{\text{diff}}$ has a single initial state q_0, which is also the only accepting state. It processes the word w in subwords $\#^2a^k\#a^m\#$ in the following way. First, it reads $\#^2$ upon which it takes transitions from q_0 to q_1 and from q_1 to q_2, where it starts counters 1 and 2. Next, it moves to the state q_2 where it counts letters a incrementing counter 1 and decrementing counter 2. Then, upon reading $\#$, it moves to q_3, where it counts letters a, but it decrements counter 1 and increments counter 2. After reading $\#^2a^k\#a^m$ the value of counter 1 is $k-m$ and counter 2 is $m-k$. In the following transition from q_3 to q_0, the automaton terminates both counters. The aggregating function of $\mathcal{A}_{\text{diff}}$ is Sup, thus the automaton discards the lower value, i.e., the value of $\#^2a^k\#a^m\#$ is $|k-m|$ and the automaton computes the supremum over values of all blocks. It follows that the value of $\#^2a^{m[1]}\#a^{m[2]}\#^3\dots$ is $\sup_{0\leq i}|m[2*i+1]-m[2*i+2]|$.

Example 5 ((Well-matched) average response time). Consider a system from Examples 2 and 3 and assume that it satisfies property SB1, i.e., at every position there is at most one pending request. Then, an automaton with a single monitor counter can compute the average response time (ART) property, which asks for the long-run average over requests of the number of steps between a request and the following grant. E.g. ART of the trace $(\boldsymbol{r}eq\#\#\boldsymbol{g}ra\boldsymbol{r}eq\#\boldsymbol{g}ra)^\omega$ is 1.5.

Note that ART of a word can be unbounded, whereas weighted ω-automata with the limit average value function return values which are bounded by the value of the maximal weight in the automaton. Therefore, the ART property cannot be expressed by weighted ω-automata with the limit average value function or any other value function considered in the literature. Below, we define an automaton with monitor counters, which expresses a more general property.

We consider an extension on the ART property, called the 2-ART property, which is essentially the ART property in systems with two types of requests $\boldsymbol{r}eq_1, \boldsymbol{r}eq_2$ and grants $\boldsymbol{g}ra_1, \boldsymbol{g}ra_2$. In these systems, requests are satisfied only by grants of an appropriate type. The automaton with two monitor counters depicted in Fig. 4 computes the 2-ART property.

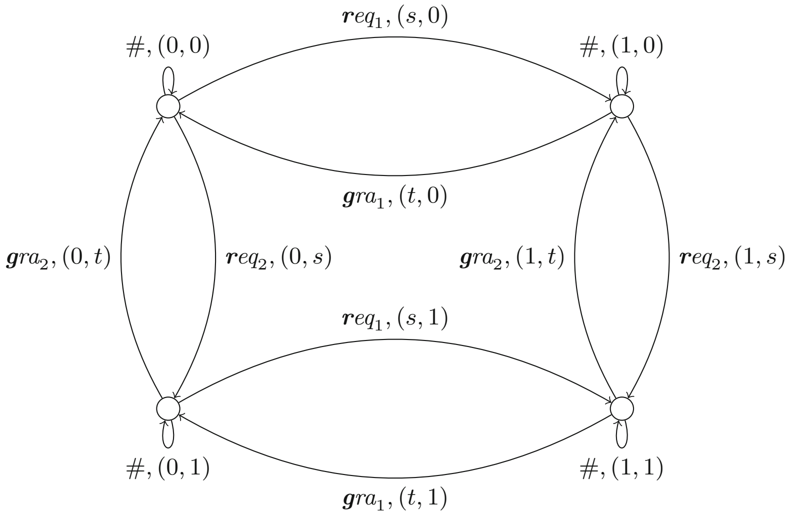
$\boldsymbol{req}_1, (s, 0)$
$\#, (0, 0)$
$\#, (1, 0)$
$\boldsymbol{gra}_1, (t, 0)$
$\boldsymbol{gra}_2, (0, t)$
$\boldsymbol{req}_2, (0, s)$
$\boldsymbol{gra}_2, (1, t)$
$\boldsymbol{req}_2, (1, s)$
$\boldsymbol{req}_1, (s, 1)$
$\#, (0, 1)$
$\#, (1, 1)$
$\boldsymbol{gra}_1, (t, 1)$

Fig. 4. The automaton computing the 2-ART property

Results. In the following section, we show equivalence of automata with monitor counters and nested weighted automata of bounded width. Due to this equivalence, we refrain from complexity discussion here.

5 Nested Weighted Automata

In this section we describe nested weighted automata introduced in [18], and closely follow the description of [18]. For more details and illustration of such automata we refer the reader to [18]. 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 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 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 works on some finite part of the input word. Once a slave automaton finishes, it returns its value to the master automaton, which treats the returned value as the weight of the current transition that is being executed. The slave automaton might immediately accept and return value $\bot$, which corresponds to *silent* transitions. If one of slave automata rejects, the nested weighted automaton rejects. In the sequel, a master automaton is always a weighted ω-automaton, and the slave automata are weighted automata. For brevity and uniformity, we simply use weighted automata. We define this formally as follows.

Nested weighted automata. A *nested weighted automaton* (NWA) $\mathbb{A}$ is a tuple $\langle \mathcal{A}_{\mathrm{mas}}; f; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$, where (1) $\mathcal{A}_{\mathrm{mas}}$, called the *master automaton*, is a $\{1, \ldots, k\}$-labeled automaton over infinite words (the labels are the indexes of automata $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$), (2) f is a value function on infinite words, called the *master value function*, and (3) $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$ 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 a corresponding slave automaton. 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 an NWA $\mathbb{A}$ on an infinite word w is an infinite sequence $(\Pi, \pi_1, \pi_2, \ldots)$ such that (1) Π is a run of $\mathcal{A}_{\mathrm{mas}}$ on w; (2) 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 word of $w[i, j]$. 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 the value of f applied to the sequence of values of runs of slave automata $\pi_1, \pi_2, \ldots$ with $\bot$ values removed. 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.

Deterministic nested weighted automata. An NWA $\mathbb{A}$ is *deterministic* if (1) the master automaton and all slave automata are deterministic, and (2) slave automata recognize prefix-free languages, i.e., languages $\mathcal{L}$ such that if $w \in \mathcal{L}$, then no proper extension of w belongs to $\mathcal{L}$. Condition (2) implies that no accepting run of a slave automaton visits an accepting state twice. Intuitively, slave automata have to accept the first time they encounter an accepting state as they will not see an accepting state again.

Example 6 (Average response time). Consider an alphabet Σ consisting of requests $\boldsymbol{r}eq$, grants $\boldsymbol{g}ra$, and null instructions #. The average response time (ART) property asks for the average number of instructions between any request and the following grant. An NWA computing the average response time is depicted in Fig. 5. At every position with letter $\boldsymbol{r}eq$ the master automaton $\mathcal{A}_{\mathrm{mas}}$ of $\mathbb{A}$ invokes the slave automaton $\mathfrak{B}_1$, which computes the number of letters from its initial position to the first following grant. The automaton $\mathfrak{B}_1$ is a $\textsc{Sum}^+$-automaton. On letters # and $\boldsymbol{g}ra$ the automaton $\mathcal{A}_{\mathrm{mas}}$ invokes the slave automaton $\mathfrak{B}_2$, which is a dummy automaton, i.e., it immediately accepts and returns no weight. Invoking such a dummy automaton corresponds to taking a silent transition. Thus, the sequence of values returned by slave automata (54311... in Fig. 5), is the sequence of response times for each request. Therefore, the averages of these values is precisely the average response time; in the NWA $\mathbb{A}$, the value function f is $\textsc{LimAvg}$. Also this property cannot be expressed by a non-nested automaton: a quantitative property is a function from words to reals, and as a function the range of non-nested $\textsc{LimAvg}$-automata is bounded, whereas the ART can have unbounded values (for details see [18]).

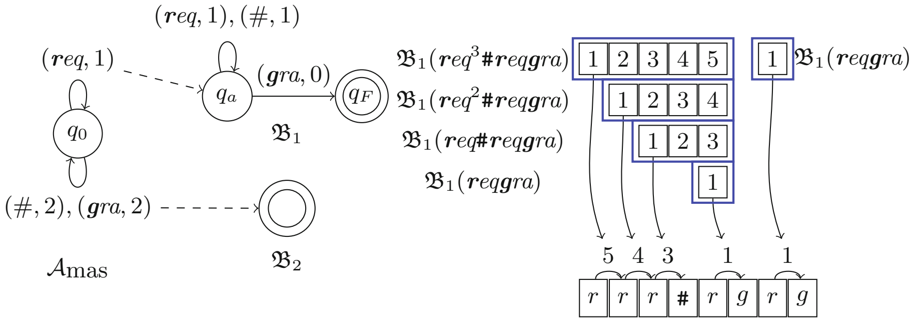

Fig. 5. An NWA $\mathbb{A}$ computing ART. The master automaton $\mathcal{A}_{\mathrm{mas}}$ and slave automata $\mathfrak{B}_1, \mathfrak{B}_2$ are on the left. A part of a run of $\mathbb{A}$ on word $\boldsymbol{r}eq\boldsymbol{r}eq\boldsymbol{r}eq\#\boldsymbol{r}eq\boldsymbol{g}ra\boldsymbol{r}eq\boldsymbol{g}ra\ldots$ is presented on the right.

5.1 NWA of Bounded Width

In this section we define a subclass of NWA, called NWA of bounded width, and we discuss properties of this subclass.

Bounded width. An NWA has *bounded width* if and only if there exists a bound C such that in every run at every position at most C slave automata are active.

Example 7 (Non-overlapping ART). Consider the NWA $\mathbb{A}$ from Example 6 depicted in Fig. 5, which does not have bounded width. The run in Fig. 5 has width at least 4, but on word $\boldsymbol{r}eq\boldsymbol{g}ra\boldsymbol{r}eq^2\boldsymbol{g}ra\boldsymbol{r}eq^3\boldsymbol{g}ra\ldots$ the number of active slave automata at position of letter $\boldsymbol{g}ra$ in subword $\boldsymbol{r}eq^i\boldsymbol{g}ra$ is i. We consider a variant of the ART property, called the 1-ART property, where after a request till it is granted additional requests are not considered. Formally, we consider the ART property over the language $\mathcal{L}_1$ defined by $(\boldsymbol{r}eq\#^*\boldsymbol{g}ra\#^*)^\omega$ (equivalently, given a request, the automata can check if the slave automaton is not active, and only then invoke it). An NWA $\mathbb{A}_1$ computing the ART property over $\mathcal{L}_1$ is obtained from the NWA from Fig. 5 by taking the product of the master automaton $\mathcal{A}_{\text{mas}}$ (from Fig. 5) with an automaton recognizing the language $\mathcal{L}_1$, which is given by the automaton for the property SB1 presented in Fig. 2. The automaton $\mathbb{A}_1$, as well as, $\mathbb{A}$ from Example 6 are $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata and they are deterministic. Indeed, the master automaton and the slave automata of $\mathbb{A}_1$ (resp., $\mathbb{A}$) are deterministic and the slave automata recognize prefix-free languages. Moreover, in any (infinite) run of $\mathbb{A}_1$ at most one slave automaton is active, i.e., $\mathbb{A}_1$ has width 1. The dummy slave automata do not increase the width as they immediately accept, and hence they are not considered as active even at the position they are invoked. Finally, observe that the 1-ART property can return unbounded values, which implies that there exists no (non-nested) $\textsc{LimAvg}$-automaton expressing it.

Lemma 1 (Translation Lemma [17]). *For every value function $f \in \mathsf{InfVal}$ on infinite words we have the following: (1) Every deterministic f-automaton with monitor counters $\mathcal{A}^{m\text{-}c}$ can be transformed in polynomial time into an equivalent deterministic $(f; \textsc{Sum})$-automaton of bounded width. (2) Every non-deterministic (resp., deterministic) $(f; \textsc{Sum})$-automaton of bounded width can be transformed in exponential time into an equivalent non-deterministic (resp., deterministic) f-automaton with monitor counters.*

Proof **(Translation of automata with monitor counters to NWA):** Consider a deterministic f-automaton $\mathcal{A}^{\text{m-c}}$ with k monitor counters and the set of states Q^{m-c} We define an $(f; \textsc{Sum})$-automaton $\mathbb{A}$, which consists of a master automaton $\mathcal{A}_{\text{mas}}$ and slave automata $\{\mathfrak{B}_{i,q} : i \in \{1,\ldots,k\}, q \in Q^{m-c}\} \cup \{\mathfrak{B}_\perp\}$. The slave automaton $\mathfrak{B}_\perp$ is a dummy automaton, i.e., it has only a single state which is both the initial and the accepting state. Invoking such an automaton is equivalent to taking a silent transition (with no weight). Next, the master automaton $\mathcal{A}_{\text{mas}}$ and slave automata $\{\mathfrak{B}_{i,q} : i \in \{1,\ldots,k\}, q \in Q^{m-c}\}$ are variants of $\mathcal{A}^{\text{m-c}}$, i.e., they share the underlying transition structure. The automaton $\mathcal{A}_{\text{mas}}$ simulates $\mathcal{A}^{\text{m-c}}$, i.e., it has the same states and the transitions among these states as $\mathcal{A}^{\text{m-c}}$. However, whenever $\mathcal{A}^{\text{m-c}}$ activates counter i, the master automaton invokes the slave automaton $\mathfrak{B}_{i,q}$, where q is their current state (both $\mathcal{A}_{\text{mas}}$ and the simulated $\mathcal{A}^{\text{m-c}}$). The accepting condition of $\mathcal{A}_{\text{mas}}$ is the same as of $\mathcal{A}^{\text{m-c}}$. For every $i \in \{1,\ldots,k\}$, the slave automaton $\mathfrak{B}_{i,q}$ keeps track of counter i, i.e., it simulates $\mathcal{A}^{\text{m-c}}$ and applies instructions of $\mathcal{A}^{\text{m-c}}$ for counter i to its value. That is, whenever $\mathcal{A}^{\text{m-c}}$ changes the value of counter i by m, the automaton $\mathfrak{B}_{i,q}$ takes a transition of the weight m. Finally, $\mathfrak{B}_{i,q}$ terminates precisely when $\mathcal{A}^{\text{m-c}}$ terminates counter i. The semantics of automata with monitor counters implies that $\mathbb{A}$ accepts if and only if $\mathcal{A}^{\text{m-c}}$ accepts and, for every word, the sequences of weights produced by the runs of $\mathbb{A}$ and $\mathcal{A}^{\text{m-c}}$ on that word coincide. Therefore, the values of $\mathbb{A}$ and $\mathcal{A}^{\text{m-c}}$ coincide on every word.

(Translation of NWA of bounded width to automata with monitor counters): We show that non-deterministic (resp., deterministic) f-automata with monitor counters subsume non-deterministic (resp., deterministic) $(f; \textsc{Sum})$-automata of bounded width. Consider a non-deterministic $(f; \textsc{Sum})$-automaton $\mathbb{A}$ with width bounded by k. We define an f-automaton $\mathcal{A}^{\text{m-c}}$ with k monitor counters that works as follows. Let Q_{mas} be the set of states of the master automaton of $\mathbb{A}$ and Q_s be the union of the sets of states of the slave automata of $\mathbb{A}$. The set of states of $\mathcal{A}^{\text{m-c}}$ is $Q_{mas} \times Q_s \times \ldots \times Q_s = Q_{mas} \times (Q_s)^k$. The automaton $\mathcal{A}^{\text{m-c}}$ simulates runs of the master automaton and slave automata by keeping track of the state of the master automaton and states of up to k active slave automata. Moreover, it uses counters to simulate the values of slave automata, i.e., whenever a slave automaton is activated, $\mathcal{A}^{\text{m-c}}$ simulates the execution of this automaton and assigns some counter i to that automaton. Next, when the simulated slave automaton takes a transition of the weight m the automaton $\mathcal{A}^{\text{m-c}}$ changes the value of counter i by m. Finally, $\mathcal{A}^{\text{m-c}}$ terminates counter i when the corresponding slave automaton terminates.

Since $\mathbb{A}$ has width bounded by k, the simulating automaton $\mathcal{A}^{\text{m-c}}$ never runs out of counters to simulate slave automata. Moreover, as it simulates runs of the master automaton and slave automata of $\mathbb{A}$, there is a one-to-one correspondence between runs of $\mathcal{A}^{\text{m-c}}$ and runs of $\mathbb{A}$ and accepting runs of $\mathbb{A}$ correspond to accepting runs of $\mathcal{A}^{\text{m-c}}$. Finally, the sequence of weights for the master automaton determined by a given run of $\mathbb{A}$ coincides with the sequence of weights of $\mathcal{A}^{\text{m-c}}$ on the corresponding run. Therefore, the values of $\mathbb{A}$ and $\mathcal{A}^{\text{m-c}}$ coincide on every word. Thus, non-deterministic f-automata with monitor counters subsume non-deterministic $(f; \textsc{Sum})$-automata of bounded width. Moreover, the one-to-one correspondence between runs of $\mathbb{A}$ and $\mathcal{A}^{\text{m-c}}$ implies that if $\mathbb{A}$ is deterministic, then $\mathcal{A}^{\text{m-c}}$ is deterministic. Therefore, deterministic f-automata with monitor counters subsume deterministic $(f; \textsc{Sum})$-automata of bounded width. This completes the proof.

Results. Complexity of the emptiness and the universality problems depends on the value functions for the master automaton and slave automata. Both problems can be undecidable. However, in decidable cases the problems are in PTime for fixed width and PSpace-complete if the width is given in input. The detailed complexity results are summarized in Tables 1–4 in [18] and Table 1 in [16].

5.2 NWA of Unbounded Width

NWA of unbounded width express the unrestricted average response time property, where there is no restriction on the number of pending requests. This property has been showed in Example 6.

Significance of results. The above example shows that NWA with unbounded width can express properties such as average response time. As compared to NWA with bounded width, analysis of NWA with unbounded width is more challenging as there is no bound on the number of active slave automata.

This model also corresponds to automata with monitor counters, where fresh counters can be activated, and there is no bound on the number of counters. Thus establishing positive results (such as decidability and complexity bounds) is much more challenging and non-trivial for NWA with unbounded width. Below we discuss the results about NWA with unbounded width.

Results. Complexity of the emptiness and the universality problems depends on the value functions for the master automaton and slave automata. Both problems can be undecidable. In decidable cases the complexity ranges between PSpace-complete and ExpSpace. The detailed complexity results are summarized in Tables 1–4 in [18].

6 Conclusion

In this work we considered various automata-theoretic formalisms to express quantitative properties. The two different extensions of weighted ω-automata with monitors, namely, with counters, or with weighted automata, have the same expressive power and provide a robust framework for quantitative specifications. There are several interesting directions of future works. First, we consider specific value functions, such as limit-average. A framework of quantitative specifications with general value functions and their characterization is an interesting direction of future work. Second, to explore the effectiveness of weighted automata with monitors in verification, such as in runtime verification, is another interesting direction of future work.

References

1. Almagor, S., Boker, U., Kupferman, O.: Discounting in LTL. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014 (ETAPS). LNCS, vol. 8413, pp. 424–439. Springer, Heidelberg (2014)
2. Alur, R., D'Antoni, L., Deshmukh, J.V., Raghothaman, M., Yuan, Y.: Regular functions and cost register automata. In: LICS 2013, pp. 13–22 (2013)
3. Baier, C., Dubslaff, C., Klüppelholz, S.: Trade-off analysis meets probabilistic model checking. In: CSL-LICS 2014, pp. 1:1–1:10 (2014)
4. Baier, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Weight monitoring with linear temporal logic: complexity and decidability. In: CSL-LICS 2014, pp. 11:1–11:10 (2014)
5. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. ACM TOCL **15**(4), 1–25 (2014)
6. Bollig, B., Gastin, P., Monmege, B., Zeitoun, M.: Pebble weighted automata and transitive closure logics. In: Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G., Abramsky, S. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 587–598. Springer, Heidelberg (2010)
7. Bouyer, P., Markey, N., Matteplackel, R.M.: Averaging in LTL. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 266–280. Springer, Heidelberg (2014)

8. Brázdil, T., Brozek, V., Chatterjee, K., Forejt, V., Kucera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. In: LICS 2011, pp. 33–42 (2011)
9. Brázdil, T., Chatterjee, K., Forejt, V., Kucera, A.: Multigain: a controller synthesis tool for MDPs with multiple mean-payoff objectives. In: TACAS 2015, pp. 181–187 (2015)
10. Chatterjee, K.: Markov decision processes with multiple long-run average objectives. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 473–484. Springer, Heidelberg (2007)
11. Chatterjee, K., Doyen, L.: Energy and mean-payoff parity Markov decision processes. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 206–218. Springer, Heidelberg (2011)
12. Chatterjee, K., Doyen, L., Edelsbrunner, H., Henzinger, T.A., Rannou, P.: Mean-payoff automaton expressions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 269–283. Springer, Heidelberg (2010)
13. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. LMCS, 6(3) (2010)
14. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM TOCL **11**(4), 23 (2010)
15. Chatterjee, K., Forejt, V., Wojtczak, D.: Multi-objective discounted reward verification in graphs and MDPs. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 228–242. Springer, Heidelberg (2013)
16. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted limit-average automata of bounded width. To appear at MFCS 2016 (2016)
17. Chatterjee, K., Henzinger, T.A., Otop, J.: Quantitative automata under probabilistic semantics. To appear at LICS 2016 (2016)
18. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted automata. In: LICS 2015, pp. 725–737 (2015)
19. Chatterjee, K., Komárková, Z., Kretínský, J.: Unifying two views on multiple mean-payoff objectives in Markov Decision Processes. In: LICS 2015, pp. 244–256 (2015)
20. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Markov decision processes with multiple objectives. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 325–336. Springer, Heidelberg (2006)
21. Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., Toruńczyk, S.: Energy and mean-payoff games with imperfect information. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 260–274. Springer, Heidelberg (2010)
22. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata, 1st edn. Springer, Heidelberg (2009)
23. Droste, M., Rahonis, G.: Weighted automata and weighted logics on infinite words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 49–58. Springer, Heidelberg (2006)
24. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, New York (1996)
25. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 112–127. Springer, Heidelberg (2011)
26. Pnueli, A., The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
27. Puterman, M.L., Processes, M.D.: Discrete Stochastic Dynamic Programming, 1st edn. Wiley, New York (1994)
28. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, vol. b, pp. 133–191. MIT Press, Cambridge (1990)