QuAK: Quantitative Automata Kit*

Marek Chalupa[1], Thomas A. Henzinger[1], Nicolas Mazzocchi[2,1]**, and N. Ege Saraç[1]***

[1] Institute of Science and Technology Austria (ISTA), Austria
{mchalupa,tah,esarac}@ista.ac.at
[2] Slovak University of Technology in Bratislava, Slovak Republic
nicolas.mazzocchi@stuba.sk

Abstract. System behaviors are traditionally evaluated through binary classifications of correctness, which do not suffice for properties involving quantitative aspects of systems and executions. Quantitative automata offer a more nuanced approach, mapping each execution to a real number by incorporating weighted transitions and value functions generalizing acceptance conditions. In this paper, we introduce QuAK, the first tool designed to automate the analysis of quantitative automata. QuAK currently supports a variety of quantitative automaton types, including Inf, Sup, LimInf, LimSup, LimInfAvg, and LimSupAvg automata, and implements decision procedures for problems such as emptiness, universality, inclusion, equivalence, as well as for checking whether an automaton is safe, live, or constant. Additionally, QuAK is able to compute extremal values when possible, construct safety-liveness decompositions, and monitor system behaviors. We demonstrate the effectiveness of QuAK through experiments focusing on the inclusion, constant-function check, and monitoring problems.

Keywords: quantitative safety · quantitative liveness · quantitative automata

1 Introduction

System behaviors are traditionally seen as sequences of system events, and specifications typically categorize them as correct or incorrect without providing more detailed information. This binary perspective has long been the cornerstone of formal verification. However, many interesting system properties require moving beyond this view to systematically reason about timing constraints, uncertainty, resource consumption, robustness, and more, which necessitates a more nuanced approach to the specification, modeling, and analysis of computer systems.

Quantitative automata [12] extend standard boolean ω-automata with weighted transitions and a value function that accumulates an infinite sequence of weights

* This work was supported in part by the ERC-2020-AdG 101020093.
** Corresponding author. N. Mazzocchi was affiliated with ISTA when his collaboration started.
*** Corresponding author.

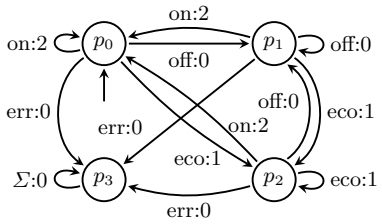


Fig. 1. A quantitative automaton $\mathcal{A}$ modeling the power consumption of a device. Associating with $\mathcal{A}$ different value functions, we can specify different aspects of its power consumption, e.g., considering LimInfAvg, the automaton specifies the long-term average power consumption.

into a single value, which generalizes the notion of acceptance condition. The common value functions include Inf, Sup, LimInf, and LimSup (respectively generalizing safety, reachability, co-Büchi and Büchi acceptance conditions), as well as DSum (discounted sum), LimInfAvg and LimSupAvg (limit average a.k.a. mean payoff). Let us consider the quantitative automaton $\mathcal{A}$ given in Figure 1, which models the power consumption of a device. With the value function Inf, it maps each execution to its minimal power consumption, whereas with LimInfAvg or LimSupAvg to its long-term average power consumption. For example, the infinite execution $(\text{off} \cdot \text{on})^\omega$ is mapped to 0 with the value function Inf, and to 1 with LimInfAvg or LimSupAvg.

The basic decision problems for boolean automata extend to this model naturally. A quantitative automaton $\mathcal{A}$ is non-empty (resp. universal) with respect to a rational number v iff $\mathcal{A}$ maps some (resp. every) infinite word w to a value at least v [12]. Notice that problem of non-emptiness (resp. universality) is closely related with computing the maximal (resp. minimal) value achievable by the automaton. As an example, consider the quantitative automaton given in Figure 1 with the value function LimInfAvg and the threshold value $v = 1$. Evidently, the maximal value achievable by $\mathcal{A}$ in Figure 1—its so-called top value—is 2, as witnessed by the word on^ω, and thus $\mathcal{A}$ is not empty with respect to $v = 1$. Moreover, an automaton $\mathcal{A}$ is included in (resp. equivalent to) another automaton $\mathcal{B}$ iff, for every infinite word w, the value mapped to w by $\mathcal{A}$ is at most (resp. exactly) the value mapped to w by $\mathcal{B}$ [12]. Beyond their close relation to game theory, these problems are relevant and interesting because solving them allows us to reason about quantitative aspects of systems and their executions.

Introduced around fifteen years ago, the model of quantitative automata has been a significant focus of research [7,9,32,33,6], with its value functions such as discounted sum and mean payoff being extensively explored in games with quantitative objectives for an even longer period [37,20]. Nonetheless, the notions of monitorability, safety, and liveness for quantitative properties have been introduced and studied only recently [25,23,24,8]. These notions and the corresponding problems are important both from a theoretical and a practical perspective, because the study of these problems deepens our understanding of

the models of quantitative automata and games, and their solutions can support our verification efforts. As for boolean properties [1,26], the ability to check if an automaton is safe lead to specialized verification techniques, e.g., the characterization of safety as continuity may enable verification techniques in domains outside of automata theory [8]. Despite the potential implications of these results on the practical verification of quantitative properties, no general tool exists for the analysis of quantitative automata.

In this paper, we introduce QuAK, the first general tool designed to automate the analysis of quantitative automata. QuAK supports a range of quantitative automaton types, including Inf, Sup, LimInf, LimSup, LimInfAvg, and LimSupAvg automata. It implements decision procedures for fundamental problems such as emptiness, universality, inclusion, equivalence, constant-check, safety, and liveness. In particular, it leverages the antichain-based inclusion algorithm for Büchi automata [18] by extending the tool FORKLIFT [17]. Additionally, QuAK provides capabilities for computing top and bottom values of quantitative automata, performing safety-liveness decompositions, and monitoring system behaviors. QuAK is designed to be free of dependencies beyond the C++ standard library and features an intuitive interface, making it accessible and easy to use.

Modeling beyond-boolean aspects of systems has been considered in several different ways. One approach considers multi-valued truth domains instead of binary domains [10,13]. Another prominent approach involves weighted automata [36], which extend classical automata by assigning each transition a numerical weight from a semiring whose operations describe how the weights are accumulated. Tools such as Vaucanson [30], Vcsn [15], and Awali [29] provide support for the analysis of weighted automata. The well-established techniques for weighted automata on finite words do not adapt well to the ω-valuation monoid framework necessary for infinite words. Quantitative automata provide a more intuitive alternative, as they are designed to generalize boolean finite-state ω-automata. See [5] for more on the distinction between weighted and quantitative automata. Another significant approach considers the interaction of digital computational processes with analog physical processes, modeled using automata [2,21] as well as temporal logics [3] and implemented in tools such as UPPAAL [28] and HyTech [22]. Signal temporal logic [31], in particular, has quantitative semantics that allows for reasoning about the degree to which a specification is satisfied or violated, and is implemented in tools such as Breach [16], S-TaLiRo [4], and RTAMT [35]. Finally, probabilistic verification deals with systems that have inherent uncertainties, such as random failures or probabilistic decision making. PRISM [27] is a widely used tool that allows for the analysis of probabilistic models like Markov chains and Markov decision processes.

2 Quantitative Properties and Automata

Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet of letters. An infinite (resp. finite) word (trace) is an infinite (resp. finite) sequence of letters $w \in \Sigma^\omega$ (resp. $u \in \Sigma^*$).

Moreover, we denote by Σ^+ the set of non-empty finite words. Given $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we write $u \prec w$ (resp. $u \preceq w$) when u is a strict (resp. nonstrict) prefix of w. We denote by $|w|$ the length of $w \in \Sigma^* \cup \Sigma^\omega$ and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w. For $w \in \Sigma^* \cup \Sigma^\omega$ and $0 \leq i < |w|$, we denote by $w[i]$ the ith letter of w. An infinite word is *ultimately periodic* (a.k.a, a *lasso word*) iff it is of the form uv^ω for some $u, v \in \Sigma^*$ with $|v| > 0$.

A *value domain* $\mathbb{D}$ is a poset. We assume that $\mathbb{D}$ is a nontrivial (i.e., $\bot \neq \top$) complete lattice. Whenever appropriate, we write 0 or $-\infty$ instead of $\bot$ for the least element $\inf \mathbb{D}$, and 1 or ∞ instead of $\top$ for the greatest element $\sup \mathbb{D}$.

A *quantitative property* is a total function $\Phi : \Sigma^\omega \to \mathbb{D}$ from the set of infinite words to a value domain. A boolean property $P \subseteq \Sigma^\omega$ is a set of infinite words.

A *nondeterministic quantitative automaton* on words [12] is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; $\iota \in Q$ is the initial state; and $\delta \colon Q \times \Sigma \to 2^{\mathbb{Q} \times Q}$ is a finite transition function over weight-state pairs. A *transition* is a tuple $(q, \sigma, x, q') \in Q \times \Sigma \times \mathbb{Q} \times Q$ such that $(x, q') \in \delta(q, \sigma)$, also written $q \xrightarrow{\sigma:x} q'$. We write $\gamma(t) = x$ for the weight of a transition $t = (q, \sigma, x, q')$. An automaton $\mathcal{A}$ is deterministic if, for all $q \in Q$ and $a \in \Sigma$, the set $\delta(q, a)$ is a singleton. We require the automaton $\mathcal{A}$ to be *total* (a.k.a. *complete*), meaning that for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{\sigma:x} q'$. For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton derived from $\mathcal{A}$ by setting its initial state ι to q.

A run of $\mathcal{A}$ on a word w is a sequence $\rho = q_0 \xrightarrow{w[0]:x_0} q_1 \xrightarrow{w[1]:x_1} q_2 \dots$ of transitions where $q_0 = \iota$ and $(x_i, q_{i+1}) \in \delta(q_i, w[i])$. When w is a finite word, we also write $\rho = q_0 \xrightarrow{w}_{\mathcal{A}} q'$ as shorthand, where q' one of the states $\mathcal{A}$ reaches after reading w. For $0 \leq i < |w|$, we denote the ith transition in ρ by $\rho[i]$, and the finite prefix of ρ up to and including the ith transition by $\rho[..i]$. Since each transition t_i carries a weight $\gamma(t_i) \in \mathbb{Q}$, the sequence ρ provides a weight sequence $\gamma(\rho) = \gamma(t_0)\gamma(t_1)\dots$. A Val-automaton is equipped with a value function $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$, which assigns real values to runs of $\mathcal{A}$. The value of a run ρ is $\mathsf{Val}(\gamma(\rho))$. The value of a Val-automaton $\mathcal{A}$ on a word w, denoted $\mathcal{A}(w)$, is the supremum of $\mathsf{Val}(\rho)$ over all runs ρ of $\mathcal{A}$ on w. The *top value* of a Val-automaton $\mathcal{A}$, denoted $\top_{\mathcal{A}}$ (or $\top$ when $\mathcal{A}$ is clear from the context), is the supremum of $\mathcal{A}(w)$ over all words w. Similarly, the *bottom value* of $\mathcal{A}$, denoted $\bot_{\mathcal{A}}$ (or $\bot$), is the infimum of $\mathcal{A}(w)$ over all words w.

We list below the common value functions for quantitative automata, defined over infinite sequences $v_0 v_1 \dots$ of rational weights.

- $\mathsf{Inf}(v) = \inf\{v_n \mid n \geq 0\}$
- $\mathsf{Sup}(v) = \sup\{v_n \mid n \geq 0\}$
- $\mathsf{LimInf}(v) = \lim_{n\to\infty} \inf\{v_i \mid i \geq n\}$
- $\mathsf{LimSup}(v) = \lim_{n\to\infty} \sup\{v_i \mid i \geq n\}$
- $\mathsf{LimInfAvg}(v) = \mathsf{LimInf}\left(\frac{1}{n}\sum_{i=0}^{n-1} v_i\right)$
- $\mathsf{LimSupAvg}(v) = \mathsf{LimSup}\left(\frac{1}{n}\sum_{i=0}^{n-1} v_i\right)$
- For a discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$, $\mathsf{DSum}_\lambda(v) = \sum_{i\geq 0} \lambda^i v_i$

All of these classes of automata are sup-closed [8, Prop. 2.2]: for every finite word $u \in \Sigma^*$ there is a continuation $w \in \Sigma^\omega$ with $\mathcal{A}(uw) = \sup_{w' \in \Sigma^\omega} \mathcal{A}(uw')$.

2.1 Basic Decision Problems of Quantitative Automata

Non-emptiness An automaton $\mathcal{A}$ is *non-empty* with respect to a threshold $v \in \mathbb{Q}$ iff $\mathcal{A}(w) \geq v$ for some $w \in \Sigma^\omega$. Since the top value of an automaton is achievable by a lasso word [12, Thm. 3], it is easy to see that $\mathcal{A}$ is non-empty with respect to v iff $\top_\mathcal{A} \geq v$. The top value of the common classes of quantitative automata can be computed as follows: For Inf and Sup automata, the top value can be computed by a simple extension of the standard attractor construction; for LimInf and LimSup automata, by a simple extension of the standard recurrence construction. For LimInfAvg and LimSupAvg automata, it can be computed by Karp's maximum mean cycle algorithm; for DSum automata, by solving a game with discounted payoff objectives in graphs with rewards on edges. All of these computations are in PTime, therefore the non-emptiness problem is in PTime for these automata classes.

Universality An automaton $\mathcal{A}$ is *universal* with respect to a threshold $v \in \mathbb{Q}$ iff $\mathcal{A}(w) \geq v$ for all $w \in \Sigma^\omega$. For Inf, Sup, LimInf, and LimSup automata, the universality problem is PSpace-complete [12, Thm. 7]. This is achieved by a simple reduction to the boolean universality problem (of safety, reachability, co-Büchi, and Büchi automata, respectively). For nondeterministic LimInfAvg and LimSupAvg automata, the problem is undecidable [14,11], and for nondeterministic DSum automata, it is a long-standing open problem. Nonetheless, for their deterministic counterparts the problem is in PTime by a simple product construction [12, Thm. 8].

Notice that an automaton is universal with respect to a threshold v iff its bottom value $\bot$ is at least v. For a deterministic automaton $\mathcal{A}$, the bottom value can be computed in PTime as $\bot_\mathcal{A} = -\top_{\hat{\mathcal{A}}}$ where $\hat{\mathcal{A}}$ is a copy of $\mathcal{A}$ in which all the weights are multiplied by -1 and the value function is replaced by its dual (e.g., using Inf instead of Sup). For a nondeterministic Inf, Sup, LimInf, or LimSup automaton $\mathcal{A}$, we can compute its bottom value via repeated inclusion checks against a constant automaton: the largest weight x of $\mathcal{A}$ such that the constant automaton $\mathcal{B}$ with the value x is included in $\mathcal{A}$ equals $\bot_\mathcal{A}$, which gives us a PSpace algorithm.

Inclusion An automaton $\mathcal{A}$ is *included* in another automaton $\mathcal{B}$ iff $\mathcal{A}(w) \leq \mathcal{B}(w)$ for all $w \in \Sigma^\omega$. Like the universality problem, the inclusion is PSpace-complete for Inf, Sup, LimInf, and LimSup automata [12, Thm. 4]. As expected, for nondeterministic LimInfAvg and LimSupAvg automata, it is again undecidable (by a reduction from universality), and for nondeterministic DSum automata, it is again open. When $\mathcal{B}$ is deterministic, the problem is again in PTime by a simple product construction [12, Thm. 5]. Analogously, $\mathcal{A}$ is *equivalent* to $\mathcal{B}$ iff

$\mathcal{A}(w) = \mathcal{B}(w)$ for all $w \in \Sigma^\omega$. The decidability and complexity results for the inclusion problem carry to equivalence problem.

The standard approach for Inf, Sup, LimInf, and LimSup automata relies on reducing the problem to boolean inclusion: an automaton $\mathcal{A}$ is included in another automaton $\mathcal{B}$ iff $L(\mathcal{A}, v) \subseteq L(\mathcal{B}, v)$ holds for every weight v of $\mathcal{A}$, where $L(\mathcal{A}, v) = \{w \in \Sigma^\omega \mid \mathcal{A}(w) \geq v\}$. QuAK takes an alternative approach based on antichains [38,18], which we detail in Section 3.

2.2 Safety and Liveness of Quantitative Automata

Constant-Function Problem A quantitative automaton $\mathcal{A}$ defines a constant function iff there exists $c \in \mathbb{R}$ such that $\mathcal{A}(w) = c$ for all $w \in \Sigma^\omega$. For all common classes of quantitative automata, deciding whether they define a constant function is PSPACE-complete [8, Prop. 3.2, Thms. 3.3 and 3.7]. Deciding whether an automaton defines a constant function is closely related with deciding its safety and liveness [8]. As we will discuss below, this is especially important for limit average automata whose equivalence is undecidable and for discounted sum automata whose universality is open. For limit average automata, the constant-function problem can be solved by a reduction to the limitedness problem of distance automata. For discounted sum automata, the problem is reduced to the universality problem of nondeterministic finite automata on finite words. For the remaining classes of automata, one can simply check if the given automaton is universal with respect to its top value.

Safety The boolean membership problems asks, given a boolean property $P \subseteq \Sigma^\omega$ and a word $w \in \Sigma^\omega$, whether w belongs to P. Then, a boolean property P is safe iff every word w that is not a member of P has a finite prefix $u \prec w$ such that for every continuation w' the word uw' is not a member of P. The quantitative decision problems discussed above implicitly generalize the boolean membership problem to the quantitative setting by asking, given an automaton $\mathcal{A}$, a value v, and a word w, whether the value $\mathcal{A}(w)$ is at least v. The quantitative generalization of safety reflects this view: a quantitative property Φ is safe iff every wrong membership query has a finite witness for the violation.

Definition 1 ([24]). *A quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* safe *iff for every $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ with $\Phi(w) \not\geq v$, there exists a finite prefix $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$.*

The safety closure $\mathit{SafetyCl}(\Phi)$ of a property Φ ensures its safety by minimally increasing the value of each word, and a property is safe iff it is equal to its safety closure (see [24, Defn. 5, Prop. 6, Thm. 9]). For the common classes of quantitative automata, we can compute their safety closure in PTIME by assigning the top value of each state to all its incoming transitions [8, Thm. 4.18].

The decision procedures for safety in quantitative automata depend on the value functions, with Inf and DSum automata always defining safety properties [8, Thm. 4.15], while for Sup, LimInf, and LimSup automata, safety check

is PSPACE-complete thanks to the decidability of their equivalence [8, Thm. 4.22]. For LimInfAvg and LimSupAvg automata, despite the undecidability of their equivalence problem, their safety can be decided in EXPSPACE by using their constant-function check as a subroutine [8, Thm. 4.23].

Liveness As for boolean safety, the notion of boolean liveness takes the membership problem to its basis. A boolean property P is live iff for every finite word u (even if u is prefix of some word that is not a member of P) there is a continuation w such that the word uw is a member of P. Quantitative liveness extends this membership-based view: a quantitative property Φ is live iff, whenever a property value is less than $\top$, there exists a value v for which the wrong membership query $\Phi(w) \geq v$ can never be dismissed by any finite witness $u \prec w$.

Definition 2 ([24]). *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* live *when for all $w \in \Sigma^\omega$, if $\Phi(w) < \top$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(w) \not\geq v$ and for all prefixes $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$.*

In the quantitative setting, liveness is characterized by the safety closure operation, where a quantitative property Φ is live iff $\Phi(w) < \mathit{SafetyCl}(\Phi)(w)$ for all words w with $\Phi(w) < \top$ [24, Thm. 37]. For sup-closed quantitative properties (like those defined by quantitative automata thanks to [8, Prop. 2.2]), a property is live iff its safety closure defines a constant function $\top$ [8, Thm. 5.7].

To decide liveness for a quantitative automaton $\mathcal{A}$, one can check if $\mathit{SafetyCl}(\mathcal{A})$ maps all words w to $\top$, which involves checking the universality of $\mathit{SafetyCl}(\mathcal{A})$ with respect to $\top$. Note that $\mathit{SafetyCl}(\mathcal{A})$ is an Inf automaton when $\mathsf{Val} \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$, and a DSum automaton when $\mathsf{Val} = \mathsf{DSum}$. For Inf automata, this problem is PSPACE-complete. For DSum automata, despite their universality is open, checking whether their liveness check is PSPACE-complete [8, Thm. 5.9] since checking whether they define a constant function is PSPACE-complete.

Safety-Liveness Decompositions Every boolean property P is the intersection of a boolean safety property S and a boolean liveness property L. In this decomposition, S is the boolean safety closure of P, while L is the union of P with the complement of S. In a way, the liveness part L balances the safety closure S by not including the words that belong to the difference of S and P.

Every quantitative property Φ is the pointwise minimum of its safety closure $\mathit{SafetyCl}(\Phi)$ and a liveness property Ψ [24, Thm. 44]. In particular, the liveness component Ψ is defined similarly to its boolean analogue: for every word w, we have $\Psi(w) = \top$ if $\Phi(w) = \mathit{SafetyCl}(\Phi)(w)$, and $\Psi(w) = \Phi(w)$ if $\Phi(w) < \mathit{SafetyCl}(\Phi)(w)$. For deterministic Sup, LimInf, and LimSup automata, the decomposition works the same way on the level of transition weights (see [8, Thms. 5.10 and 5.11]), which is achievable in PTIME. Note that Sup and LimInf automata are determinizable [12, Thm. 13].

3 The Tool

QuAK implements the decision procedures and constructions described in Sections 2.1 and 2.2. In particular, let $\mathcal{A}$ and $\mathcal{B}$ be Val automata where $\mathsf{Val} \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$, and let $v \in \mathbb{Q}$ be a rational number. QuAK currently supports the following operations (whenever known to be computable):

1. Check if $\mathcal{A}$ is non-empty with respect to v.
2. Check if $\mathcal{A}$ is universal with respect to v.
3. Check if $\mathcal{A}$ is included in $\mathcal{B}$.
4. Check if $\mathcal{A}$ defines a constant function.
5. Check if $\mathcal{A}$ defines a safety property.
6. Check if $\mathcal{A}$ defines a liveness property.
7. Compute the top value $\top$ of $\mathcal{A}$.
8. Compute the bottom value $\bot$ of $\mathcal{A}$.
9. Compute the safety closure of $\mathcal{A}$.
10. Compute the safety-liveness decomposition of $\mathcal{A}$.
11. Construct and execute a monitor for $\mathcal{A}$.

The tool is written in C++ using the standard library, and is available at https://github.com/ista-vamos/QuAK together with detailed instructions on its usage. In this section, we describe the automata representation, our antichain-based inclusion algorithm, implementation of the constant-function check for limit average automata, and monitoring approach for quantitative properties. The rest follows the descriptions in Sections 2.1 and 2.2.

Automata Representation We represent automata in a way that makes the algorithms as efficient as possible while keeping their implementation convenient and maintainable. Here, we explain some choices we made for this sake.

Automata objects do not have a value function because it may be useful to interpret the same transition structure in different ways (like in Figure 1). The user needs to specify the value function when a decision procedure or a construction is called on an automaton. Moreover, each automaton contains a directed acyclic graph representing its strongly connected components (SCCs), which is constructed once the automaton is created. Each state has a tag representing the SCC it belongs to. Moreover, in addition to storing the outgoing transitions of a state, we also store the incoming transitions. These are useful when computing the top value, constructing the safety closure, and determinizing the safety closure of limit average automata (for deciding their safety). Finally, while boolean automata has a fixed domain $\{0, 1\}$, each quantitative automaton may define a different domain. To address this, each automaton has two numerical variables representing the minimum and maximum of its domain, whose values are taken as the minimum and maximum of the automaton's weights by default.

Antichain-based Inclusion Algorithm The language inclusion of Büchi automata (a.k.a. LimSup) is known to be PSPACE-complete. Algorithms that behave well in practice have been investigated for decades and remains an active research field. Among others, FORKLIFT [17] uses the Ramsey-based technique and leverages the antichains heuristic. The algorithm to decide whether $L(A) \subseteq L(B)$ holds searches counterexamples and runs membership queries. The Ramsey-based approach prunes the search for inclusion violation by discarding candidates of $L(A)$ which are subsumed by others words of $L(A)$ with respect to a given well-quasiorder. Termination comes from the mathematical properties of well-quasiorders guaranteeing that only finitely many candidates will be kept after pruning. Correctness is trivial: if a kept candidate violates $L(A) \subseteq L(B)$ then the inclusion does not holds. To guarantee completeness, we require the quasiorder to fulfill, for all candidate $w \in \Sigma^\omega$ subsumed by $w_0 \in \Sigma^\omega$, that $w_0 \in L(B)$ implies $w \in L(B)$. Hence, if all candidates belong to $L(B)$ then so do the discarded ones. The antichains heuristic allows a symbolic fixpoint computation of the remaining candidates [38].

Language inclusion can be decided by reasoning solely on ultimately periodic words (a.k.a. lasso words). So, the candidates are words of the form uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$ are called a stem and a period, respectively. [19] provides an algorithm that uses two quasiorders: one for the stems and one for the periods. Since using different quasiorders yields more pruning when searching for a inclusion violation, [18] considers using an unbounded number of quasiorders called FORQ: one for the stems and a family of quasiorders for the periods each of them depending on a distinct stem. It is worth emphasizing that each quasiorder requires a fixpoint computation, and thus, the more quasiorders are handled, the more the antichains heuristic is leveraged.

The novelty of FORKLIFT lies in the use of FORQ to discard candidates. Below, we generalize FORQs for Büchi automata defined in [18] to support any LimSup automata.

Definition 3. *Let $\mathcal{B} = (\Sigma, Q, \iota, \delta)$ be a* LimSup *automaton over the weights $W = \{\gamma(t) \mid t \text{ is a transition of } \mathcal{B}\}$. The structural FORQ of $\mathcal{B}$ is the pair $(\precsim^{\mathcal{B}}, \{\precsim^{\mathcal{B}}_u\}_{u \in \Sigma^*})$ where the quasiorders are defined by:*

$$u_1 \precsim^{\mathcal{B}} u_2 \iff \mathsf{Tgt}_{\mathcal{B}}(u_1) \subseteq \mathsf{Tgt}_{\mathcal{B}}(u_2)$$
$$v_1 \precsim^{\mathcal{B}}_u v_2 \iff \mathsf{Cxt}_{\mathcal{B}}(\mathsf{Tgt}_{\mathcal{B}}(u), v_1) \subseteq \mathsf{Cxt}_{\mathcal{B}}(\mathsf{Tgt}_{\mathcal{B}}(u), v_2)$$

with $\mathsf{Tgt}_{\mathcal{B}} \colon \Sigma^* \to 2^Q$ and $\mathsf{Cxt}_{\mathcal{B}} \colon 2^Q \times \Sigma^+ \to 2^{Q \times Q \times W}$ such that

$$\mathsf{Tgt}_{\mathcal{B}}(u) = \{q' \in Q \mid \iota \xrightarrow{u}_{\mathcal{B}} q'\}$$
$$\mathsf{Cxt}_{\mathcal{B}}(S, v) = \{(q, q', x) \mid q \in S, \rho = q \xrightarrow{v}_{\mathcal{B}} q', \text{and } x \text{ is the maximum of the weight sequence of } \rho\}$$

The modification appears in the definition of $\mathsf{Cxt}_{\mathcal{B}}$ where x ranges over the weights of $\mathcal{B}$ instead of $\{\bot, \top\}$. Extending all the properties on structural FORQ

established by [18] to this definition is straightforward, and implies the soundness of our inclusion algorithm for LimSup automata. To use this algorithm for other classes of automata, we translate Inf, Sup, and LimInf automata to LimSup automata in PTime for inclusion queries.

We highlight the remaining technical modifications below.

- Given a stem $u \in \Sigma^*$, the data structure used for the fixpoint computation of $\mathsf{Cxt}_{\mathcal{B}}$ carries (as in FORKLIFT) a period $v \in \Sigma^+$, a context $\mathsf{Cxt}_{\mathcal{B}}(\mathsf{Txt}_{\mathcal{B}}(u), v)$, and (in addition to FORKLIFT) the value of $\mathcal{A}$ over uv^ω.
- In FORKLIFT, the fixpoint computation of $\mathsf{Cxt}_{\mathcal{B}}$ does not leverage SCCs. A compilation option provides an implementation for QuAK that computes $\mathsf{Cxt}_{\mathcal{B}}$ while only considering intra-SCC transitions. We call this optimization `scc-search`.

Constant-function Check for Limit Average Automata Checking whether a limit average automaton $\mathcal{A}$ is constant can be done by a reduction to the limitedness problem of distance automata [8, Thm. 3.7]. To simplify our implementation, we consider a reduction to the universality problem of LimInf automata. In essence, our reduction follows [8, Thm. 3.7] except for in two points. First, after removing negative-weighted edges by Johnson's algorithm, instead of constructing a distance automaton, we flip the weights again to construct a limit average automaton (which resolves nondeterminism by sup and has the same value function as the input automaton). This yields an automaton $\mathcal{B}$ with non-positive transition weights. Then, we construct a LimInf automaton $\mathcal{C}$ by mapping negative weights of $\mathcal{B}$ to 0, and 0-valued weights to 1. Note that it may not hold that $\mathcal{B}(w) < 0$ iff $\mathcal{C}(w) < 1$ for all words w, but since $\mathcal{C}$ recognizes an ω-regular language, we can show that $\mathcal{B}$ is constant iff $\mathcal{C}$ is universal with respect to 1.

More specifically, the reduction goes as follows. Let $\mathcal{A}$ be a LimInfAvg (resp. LimSupAvg) automaton. First, obtain $\mathcal{A}_1$ by subtracting $\top$ from all transition weights of $\mathcal{A}$. We have $\mathcal{A}_1(w) = \mathcal{A}(w) - \top$ for all words w. Then, obtain $\mathcal{A}_2$ by multiplying by -1 all transition weights of $\mathcal{A}_1$, resolving nondeterminism by inf, and taking the value function LimSupAvg (resp. LimInfAvg). We have $\mathcal{A}_2(w) = -\mathcal{A}_1(w)$ for all words w. Then, obtain $\mathcal{A}_3$ by using Johnson's algorithm to remove transitions with negative weights of $\mathcal{A}_2$. We have $\mathcal{A}_2(w) > 0$ iff $\mathcal{A}_3(w) > 0$ for all words w. Then, obtain $\mathcal{B}$ by multiplying by -1 all transition weights of $\mathcal{A}_3$, resolving nondeterminism by sup, and taking the value function LimInfAvg (resp. LimSupAvg). We have $\mathcal{B}(w) = -\mathcal{A}_3(w)$ for all words w. Note that all transitions of $\mathcal{B}$ have non-positive weights. Finally, obtain $\mathcal{C}$ by updating the weights of $\mathcal{B}$ as follows and taking the value function LimInf: if a transition has weight 0, then its new weight is 1; otherwise (weight less than 0), then its new weight is 0.

By construction, $\mathcal{A}$ is constant iff $\mathcal{B}$ is constant. We argue that $\mathcal{B}$ is constant iff $\mathcal{C}$ is universal (with respect to 1). If there is a word $u \in \Sigma^\omega$ such that $\mathcal{B}(u) < 0$, then all runs of $\mathcal{B}$ over u visit infinitely often some negative weight. Thus, $\mathcal{C}(u) < 1$ comes as a direct consequence of this implication. Note, however, that the reciprocal is not true, i.e., all runs of a word could visit infinitely often some negative weight while being mapped to 0 by $\mathcal{B}$. Now, if there is a word

$v \in \Sigma^\omega$ such that $\mathcal{C}(v) < 1$, then there exists also an ultimately periodic word $w \in \Sigma^\omega$ such that $\mathcal{C}(w) < 1$. This is because $\mathcal{C}$ is a co-Büchi automaton that defines a non-empty ω-regular language. Let w be of the form $w_1 w_2^\omega$. We define $x = |w_1|$, $y = |w_2|$, and let n be the number of states of $\mathcal{C}$. Suppose towards contradiction that some run of $\mathcal{C}$ over w visits only the weight 1 for $x + yn$ consecutive transitions. It implies that this run visits twice the same state at the end of the period w_2 while visiting only the weight 1 in between, which exhibits another run of $\mathcal{C}$ over w of value 1, and thus leads to the contradiction $\mathcal{C}(w) = 1$. Hence, all runs of $\mathcal{C}$ over w periodically visit the weight 0 after $x + yn$ transitions. Since $\mathcal{B}$ differs from $\mathcal{C}$ only in transition weights, all runs of $\mathcal{B}$ over w periodically visit some negative weight after $x + yn$ transitions, therefore $\mathcal{B}(w) < 0$. In conclusion, $\mathcal{A}$ defines a constant function iff $\mathcal{C}$ is universal (with respect to 1). Note that we can directly construct $\mathcal{C}$ from $\mathcal{A}$ in PTime.

Monitoring Given a specification represented as a deterministic quantitative automaton $\mathcal{A}$, QuAK is able create a monitor object that stores an array of top values (storing the top value of $\mathcal{A}^q$ for each state q of $\mathcal{A}$), an array of bottom values (storing the bottom value of $\mathcal{A}^q$ for each state q of $\mathcal{A}$), and a pointer to the current state of $\mathcal{A}$ (initialized as the initial state of $\mathcal{A}$). A monitor object can read input letters incrementally while getting the next state q of $\mathcal{A}$ and maintaining the lowest and highest values achievable from q, namely, the bottom and top values of $\mathcal{A}^q$. In addition, we implement running average monitors for limit average automata.

4 Experimental Evaluation

We evaluated QuAK in a set of experiments. In particular, we measure the performance our antichain-based inclusion algorithm and compare it to the standard algorithm based on repeated reduction to language inclusion of Büchi automata. These experiments include also the measurement of the impact of the `scc-search` optimization described in Section 3. Next, we evaluate the runtime of checking if an automaton defines a constant funcion. Finally, we use QuAK to runtime monitor the smoothness of a controller for a drone to show that the tool can be used in the context of quantitative runtime monitoring. The artifact to reproduce the experiments can be found at https://doi.org/10.5281/zenodo.13132069.

Setup QuAK was compiled with -O3 and link-time optimizations enabled. The `scc-search` optimization was enabled for all experiments except a part of those that aimed at evaluating this optimization (Section 4.1). All experiments ran on machines with *AMD EPYC* CPU with the frequency 3.1 GHz. The time limit was set to 100 s wall time.

Benchmarks Because of the lack of benchmarks for quantitative automata, we used randomly generated quantitative automata. All automata are complete

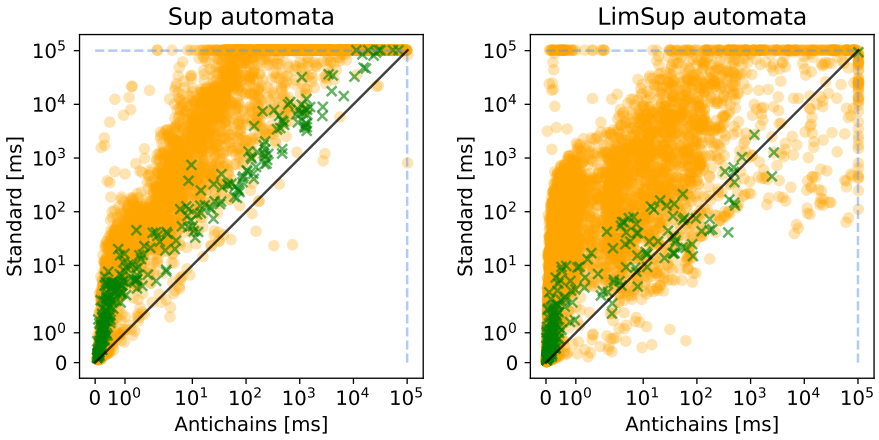


Fig. 2. CPU time in milliseconds of running *Antichains* (x axis) and *Standard* (y axis) inclusion algorithms on random automata with 2–32 states where at least one algorithm finished within time limit. The alphabet has 2. Orange dots are for pairs of automata that are not included and green crosses are for included automata. The scatter plot on the left is for Sup and on the right for LimSup value function.

(i.e., every state has an outgoing transition for each symbol in the alphabet) and have weights between -10 and 10 chosen uniformly at random. An automaton that has n states can have up to $n|\Sigma|+2n+1$ edges where Σ is the alphabet. As a result, the generated automata are non-deterministic. The number of states and the size of alphabet differ in the experiments and are always explicitely mentioned.

4.1 Comparing Inclusion Algorithms

In this subsection, we compare the standard approach to compute the quantitative automata inclusion (referred to as *Standard*) with our antichain-based inclusion algorithm (referred to as *Antichains*). The implementation of the standard approach uses the boolean version of FORKLIFT to decide the inclusion of boolean automata. Both algorithms are implemented in QuAK.

Figure 2 shows the CPU time of running *Standard* and *Antichains* algorithms for $\mathsf{Val} \in \{\mathsf{Sup}, \mathsf{LimSup}\}$. We used 100 random automata with 2–32 states and with 2-symbol alphabet. Algorithms were ran for each possible pair of the automata, which results in 10000 inclusion checks. In the plots, we show only the runs where at least one algorithm decided the inclusion.

The algorithm *Antichains* is almost always faster, often significantly, and it can finish in a lot of cases when *Standard* reaches the time limit (points on the blue dashed line). The *Standard* algorithm internally runs (the boolean version of) *Antichains* algorithm multiple times for each weight, and therefore it is expected that *Antichains* should be faster most of the times.

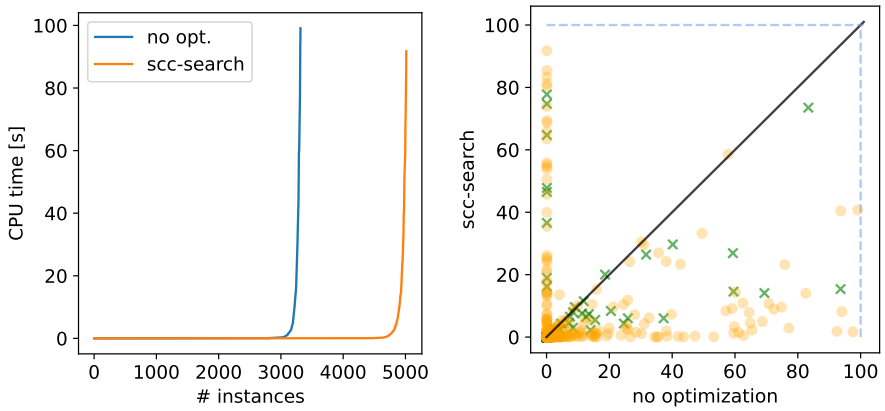


Fig. 3. CPU time of running *Antichains* algorithm with and without the `scc-search` optimization on the benchmarks from Figure 2. In the left plot, the x axis shows how many instances the inclusion algorithms are able to decide given the time limit on the y axis. The right plot compares the runtime per instance. There, orange dots are for pairs of automata that are not included and green crosses are for included automata. The plots are for Sup automata and time is in seconds.

Evaluating Optimizations of Inclusion Algorithms The results in Figure 2 are for QuAK that is compiled with the `scc-search` optimization (see Section 3). Plots in Figure 3 show that this optimization significantly improves the runtime. The plot on the left shows how many instances (the x axis) can be decided given the time limit is set to the value on the y axis. The optimization allows to decide nearly 2000 more instances in under 2 seconds. The plot on the right shows that the optimization also hurts in some cases. Nevertheless, it helps with approximately 90% of the considered automata.

4.2 Evaluating Constant-function Checking

To evaluate the constant-function checking algorithm for limit-average automata, we generated 1000 random automata with a 4-symbol alphabet and 1–100 states. The results of running the algorithm on these automata are summarized in Figure 4.

The computational complexity of the algorithm increases steeply: for larger automata, the result is typically computed either quickly or not at all. While the algorithm times out on many instances, the results suggest that deciding whether an automaton is constant remains feasible in certain cases. Notably, all instances that do not time out in our experiments fall into one of two categories: they are either deterministic, for which the problem is in PTime, or the resulting co-Büchi automaton has a very low density of accepting edges, for which the antichain-based algorithm finds witnesses for non-universality more easily.

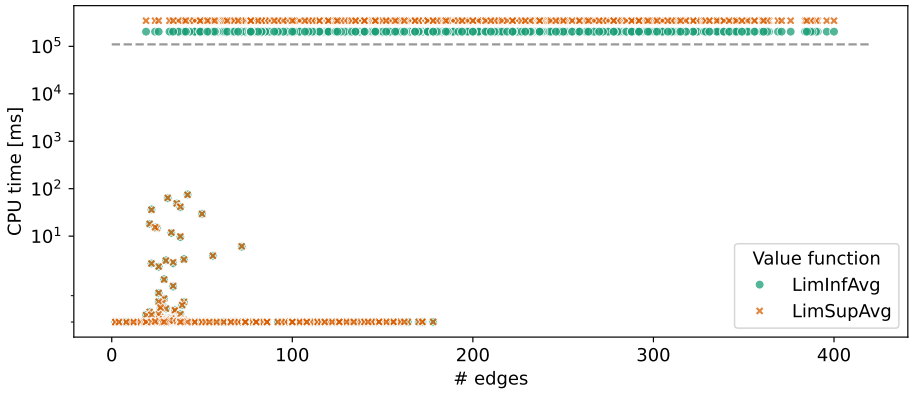


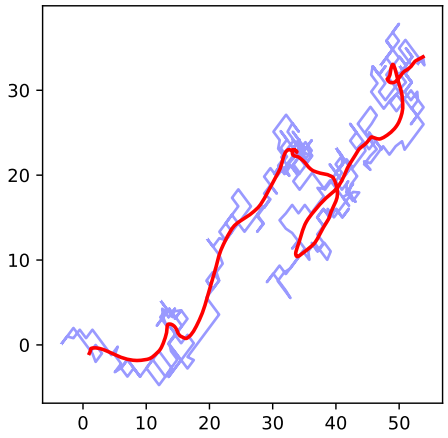
Fig. 4. CPU time of deciding if an automaton defines a constant function. Points representing timeouts were moved above the timeout line (the dashed line) and separated (with no particular order with respect to the vertical axis).

4.3 Runtime Monitoring

We experimented with using quantitative automata for runtime monitoring. Our use case is to monitor the smoothness of controllers of cyber-physical systems (CPS) [34], which means that the actions issued by a CPS controller should always cause only a relatively small change in the state of the CPS. For example, a controller of a drone should not instruct it to immediately change to the opposite of the current direction. Controllers that are not smooth can lead to increased energy consumption or even hardware failures [34].

We monitored a flying drone in a simulated environment. For simplicity, we assumed that the drone is a point with mass 1 and its energy consumption is equal to the sum of forces generated by the thrust of its engines. Each action issued by a controller is a pair of integers (x, y) that represents the acceleration vector (on a 2D plane), with $-10 \leq x, y \leq 10$. Therefore, the alphabet Σ has 441 symbols. The monitor computes the running average of weights of the automaton $\mathcal{A}_M$ that has one state for each symbol from Σ, and from each state q there is an outgoing edge $q \xrightarrow{q':x} q'$ to any other state q' under the symbol q'. In other words, the states remember the last issued action. The weight x of each transition going from q to q' is the distance between q and q'. In total, the automaton $\mathcal{A}_M$ has 441 states and 194481 edges.

The initial mission of the drone was to get from the point $(0, 0)$ to $(1000, 1000)$ (with no obstacles) using a controller that every 0.1 s issues a command to accelerate toward the target. However, a random deviation taken from the normal distribution with mean 0 and standard deviation either 1 or 5 (this is a parameter) is applied to both acceleration coordinates at every step. The magnitude of the acceleration is also random, skewed toward the maximum acceleration value



	Original		Smoothed	
Controller	*EC*	*Score*	EC	*Score*
sd=1	9.95	12.83	3.40	0.15
sd=5	12.42	15.97	2.42	0.82

Fig. 5. Results of monitoring the smoothness of a drone controller on an erratic and smoothed trajectory (*Original* and *Smoothed*, resp.). The situation is depicted on the left where the erratic trajectory is blue and the smoothed one is red. Only a part of the trajectories is shown. In the table on the right, *Score* is the value computed by the monitor and *EC* is the energy consumption of the drone on the trajectory. The lower is the score, the smoother should be the trajectory. All numbers are averages from 3 simulations.

10. If the resulting acceleration along a coordinate is greater (lower) than 10 (-10, resp.), it is set to 10 (-10, resp.).

The rather chaotic controller described above models an imperfect controller and results in navigating the drone along an erratic trajectory. We ran another mission where the drone followed the previously taken erratic trace that has been smoothed using gradient ascent. The situation is depicted on the left in Figure 5, and the results of monitoring the trajectories is on the right in the same figure. The monitor correctly assigns lower scores to smoother trajectories, which directly corresponds to the difference in energy consumption (EC).

5 Conclusion

We presented QuAK: the first software tool to automate the analysis of quantitative automata. QuAK implements several standard decision procedures as well as an antichain-based inclusion checking, algorithms to decide whether an automaton is safe, live, and constant, and a construction for its safety-liveness decomposition. In the future, we plan to add algorithms for discounted sum automata and implement safety-liveness decompositions for more classes of automata. One can also extend the tool with a support for probabilistic and nested variants of quantitative automata.

References

1. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distributed Comput. **2**(3), 117–126 (1987). https://doi.org/10.1007/BF01782772
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
3. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. Inf. Comput. **104**(1), 35–77 (1993). https://doi.org/10.1006/INCO.1993.1025
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6605, pp. 254–257. Springer (2011). https://doi.org/10.1007/978-3-642-19835-9_21
5. Boker, U.: Quantitative vs. weighted automata. In: Proc. of Reachbility Problems. pp. 1–16 (2021)
6. Boker, U.: Discounted-sum automata with real-valued discount factors. In: Sobocinski, P., Lago, U.D., Esparza, J. (eds.) Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024. pp. 15:1–15:14. ACM (2024). https://doi.org/10.1145/3661814.3662090
7. Boker, U., Henzinger, T.A.: Exact and approximate determinization of discounted-sum automata. Log. Methods Comput. Sci. **10**(1) (2014). https://doi.org/10.2168/LMCS-10(1:10)2014
8. Boker, U., Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Safety and liveness of quantitative automata. In: Pérez, G.A., Raskin, J. (eds.) 34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium. LIPIcs, vol. 279, pp. 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.CONCUR.2023.17
9. Boker, U., Henzinger, T.A., Otop, J.: The target discounted-sum problem. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015. pp. 750–761. IEEE Computer Society (2015). https://doi.org/10.1109/LICS.2015.74
10. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Halbwachs, N., Peled, D.A. (eds.) Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1633, pp. 274–287. Springer (1999). https://doi.org/10.1007/3-540-48683-6_25
11. Chatterjee, K., Doyen, L., Edelsbrunner, H., Henzinger, T.A., Rannou, P.: Mean-payoff automaton expressions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6269, pp. 269–283. Springer (2010). https://doi.org/10.1007/978-3-642-15375-4_19
12. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. **11**(4), 23:1–23:38 (2010). https://doi.org/10.1145/1805950.1805953
13. Chechik, M., Gurfinkel, A., Devereux, B.: chi-chek: A multi-valued model-checker. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceed-

ings. Lecture Notes in Computer Science, vol. 2404, pp. 505–509. Springer (2002). https://doi.org/10.1007/3-540-45657-0_41
14. Degorre, A., Doyen, L., Gentilini, R., Raskin, J., Torunczyk, S.: Energy and mean-payoff games with imperfect information. In: Dawar, A., Veith, H. (eds.) Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6247, pp. 260–274. Springer (2010). https://doi.org/10.1007/978-3-642-15205-4_22
15. Demaille, A., Duret-Lutz, A., Lombardy, S., Sakarovitch, J.: Implementation concepts in vaucanson 2. In: Konstantinidis, S. (ed.) Implementation and Application of Automata - 18th International Conference, CIAA 2013, Halifax, NS, Canada, July 16-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7982, pp. 122–133. Springer (2013). https://doi.org/10.1007/978-3-642-39274-0_12
16. Donzé, A.: Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6174, pp. 167–170. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_17
17. Doveri, K., Ganty, P., Mazzocchi, N.: FORKLIFT (v1.0). Zenodo (2022). https://doi.org/10.5281/zenodo.6552870, maintained at https://github.com/Mazzocchi/FORKLIFT
18. Doveri, K., Ganty, P., Mazzocchi, N.: Forq-based language inclusion formal testing. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 109–129. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_6
19. Doveri, K., Ganty, P., Parolini, F., Ranzato, F.: Inclusion testing of büchi automata based on well-quasiorders. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference. LIPIcs, vol. 203, pp. 3:1–3:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.CONCUR.2021.3
20. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory **8**, 109–113 (1979)
21. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996. pp. 278–292. IEEE Computer Society (1996). https://doi.org/10.1109/LICS.1996.561342
22. Henzinger, T.A., Ho, P.: HYTECH: the cornell hybrid technology tool. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S. (eds.) Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994. Lecture Notes in Computer Science, vol. 999, pp. 265–293. Springer (1994). https://doi.org/10.1007/3-540-60472-3_14
23. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Abstract monitors for quantitative specifications. In: Dang, T., Stolz, V. (eds.) Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13498, pp. 200–220. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_11
24. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Quantitative safety and liveness. In: Kupferman, O., Sobocinski, P. (eds.) Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held

as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13992, pp. 349–370. Springer (2023). https://doi.org/10.1007/978-3-031-30829-1_17
25. Henzinger, T.A., Saraç, N.E.: Quantitative and approximate monitoring. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021. pp. 1–14. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470547
26. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods Syst. Des. **19**(3), 291–314 (2001). https://doi.org/10.1023/A:1011254632723
27. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J.T., Harder, U. (eds.) Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, London, UK, April 14-17, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2324, pp. 200–204. Springer (2002). https://doi.org/10.1007/3-540-46029-2_13
28. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997). https://doi.org/10.1007/S100090050010
29. Lombardy, S., Marsault, V., Sakarovitch, J.: Awali, a library for weighted automata and transducers (version 2.3) (2022), software available at http://vaucanson-project.org/Awali/2.3/
30. Lombardy, S., Poss, R., Régis-Gianas, Y., Sakarovitch, J.: Introducing VAUCANSON. In: Ibarra, O.H., Dang, Z. (eds.) Implementation and Application of Automata, 8th International Conference, CIAA 2003, Santa Barbara, California, USA, July 16-18, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2759, pp. 96–107. Springer (2003). https://doi.org/10.1007/3-540-45089-0_10
31. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3253, pp. 152–166. Springer (2004). https://doi.org/10.1007/978-3-540-30206-3_12
32. Michaliszyn, J., Otop, J.: Approximate learning of limit-average automata. In: Fokkink, W.J., van Glabbeek, R. (eds.) 30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands. LIPIcs, vol. 140, pp. 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPICS.CONCUR.2019.17
33. Michaliszyn, J., Otop, J.: Minimization of limit-average automata. In: Zhou, Z. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021. pp. 2819–2825. ijcai.org (2021). https://doi.org/10.24963/IJCAI.2021/388
34. Mysore, S., Mabsout, B., Mancuso, R., Saenko, K.: Regularizing action policies for smooth control with reinforcement learning. In: IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021. pp. 1810–1816. IEEE (2021). https://doi.org/10.1109/ICRA48506.2021.9561138, https://doi.org/10.1109/ICRA48506.2021.9561138
35. Nickovic, D., Yamaguchi, T.: RTAMT: online robustness monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23,

2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 564–571. Springer (2020). https://doi.org/10.1007/978-3-030-59152-6_34
36. Schützenberger, M.P.: On the definition of a family of automata. Inf. Control. **4**(2-3), 245–270 (1961). https://doi.org/10.1016/S0019-9958(61)80020-X
37. Shapley, L.S.: Stochastic games. Proceedings of the national academy of sciences **39**(10), 1095–1100 (1953)
38. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.: Antichains: A new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4144, pp. 17–30. Springer (2006). https://doi.org/10.1007/11817963_5