Extending QuAK with Nested Quantitative Automata

Thomas A. Henzinger[1], Nicolas Mazzocchi[2], N. Ege Saraç[3(✉)], and Harun Yılmaz[4]

[1] Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
tah@ista.ac.at

[2] Slovak University of Technology in Bratislava, Bratislava, Slovak Republic
nicolas.mazzocchi@stuba.sk

[3] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
ege.sarac@cispa.de

[4] Sabancı University, Istanbul, Turkey
harun.yilmaz@sabanciuniv.edu

Abstract. Quantitative automata (QAs) extend finite-state automata on infinite words with weighted transitions to specify quantitative system properties. However, their finite weight sets rule out properties like average response time, where response times can be arbitrarily large. Nested quantitative automata (NQAs) overcome this limitation: a parent automaton spawns child automata to compute unbounded values over finite infixes and aggregates them into a final result. Despite this expressiveness, NQAs have lacked practical tool support to date.

We close this gap by extending the Quantitative Automata Kit (QuAK), a software tool for QA analysis, to support NQAs. Our core contribution is implementing a suite of flattening procedures that reduce NQAs to QAs, leveraging QuAK's existing decision procedures. These reductions preserve the answers to threshold decision problems, while allowing users to specify properties in the more expressive NQA formalism. The tool handles all combinations of parent aggregators (including limits and averages) and child functions (extrema and monotonic or bounded summations) for which emptiness and universality are known to be decidable. Experiments on response-time and resource-consumption benchmarks demonstrate QuAK's effectiveness.

1 Introduction

Formal verification has traditionally focused on boolean properties: a system either satisfies a specification or it does not. Yet many critical requirements are inherently quantitative; we may require that a server's average workload stays above 20%, or that a process's peak memory usage stays under 1GB. Quantitative automata (QAs) [8] address this need by assigning rational weights to transitions in finite-state ω-automata and aggregating them via value functions such as

E. Darulova et al. (Eds.): CAV 2026, LNCS 16683, pp. 418–432, 2026.
https://doi.org/10.1007/978-3-032-32526-6_20

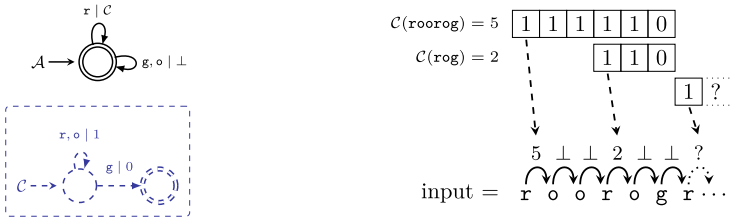

Fig. 1. A $(\mathsf{LimSupAvg}, \mathsf{Sum}^+)$-automaton $\mathbb{A} = (\mathcal{A}, \mathcal{C})$ for average response time. *Left*: Parent $\mathcal{A}$ spawns child $\mathcal{C}$ on each request. Child $\mathcal{C}$ accumulates weight 1 per step until a grant, then terminates. *Right*: On the shown prefix, the spawned instances of $\mathcal{C}$ return 5 and 2; the third instance is still active. Silent transitions (marked $\perp$) do not contribute to the parent's value, so the running average after the event `g` is $\frac{7}{2}$.

Sup, LimSup, or LimSupAvg. The Quantitative Automata Kit (QuAK) [6,7] provides the first comprehensive tool support for analyzing such automata, implementing algorithms for emptiness, inclusion, and safety-liveness analyses.

However, QAs cannot express a fundamental class of quantitative properties. Consider *average response time*: a system receives requests and issues grants, and we wish to measure the long-run average delay between each request and the following grant. For the QA value functions considered in this paper, a fixed finite transition-weight set bounds all values a QA can produce. Average response time, however, requires unbounded per-event values, since a grant may arrive arbitrarily far after its request. Therefore, no QA can express such measurements [11].

Nested quantitative automata (NQAs) [11] solve this problem. A *parent* automaton reads an infinite word and spawns *child* automata on designated transitions; each child runs over a finite infix and returns a value upon termination. The parent aggregates these values using its value function. Figure 1 shows a $(\mathsf{LimSupAvg}, \mathsf{Sum}^+)$-automaton $\mathbb{A} = (\mathcal{A}, \mathcal{C})$ that computes average response time. The parent $\mathcal{A}$ spawns the child $\mathcal{C}$ on every request; other letters do not contribute a value. The child $\mathcal{C}$ accumulates weight 1 on each step until the first grant, then terminates with the accumulated sum. On the input prefix `roorogr`$\cdots$, the first child returns 5 (waiting five steps for its grant) and the second returns 2, while the third is still running. The parent's LimSupAvg aggregates these returned values into their long-run average. Because children can run arbitrarily long, their return values are unbounded, enabling measurement of quantities that non-nested automata cannot express.

Despite solid theoretical foundations and decidability results [9–11], there has been no practical implementation of NQA algorithms. We bridge this gap by extending QuAK to support NQAs. In particular:

1. We extend QuAK with a modular architecture for NQA analysis built around *flattening*: for a given threshold decision query, QuAK translates the NQA instance into a QA instance whose yes/no answer coincides with the original query.

2. We implement flattening procedures for all emptiness and universality problems known to be decidable for combinations of parent aggregators {Sup, Inf, LimSup, LimInf, LimSupAvg, LimInfAvg} and child aggregators {Min, Max, Sum^B, Sum^+, Sum^-}.
3. We evaluate QuAK on two benchmark families—response time and resource consumption—to show its scalability across a range of NQA types and sizes.

Note that these flattening operations need not preserve semantic equivalence between the input NQA and the resulting QA, since NQAs are strictly more expressive than QAs in general, but they preserve the answer to the given threshold query. Moreover, the benchmarks reflect specifications that arise in protocol latency analysis, service-level response-time monitoring, and resource-consumption analysis for systems with dynamically started and terminated tasks. In these settings, each request, job, or process lifetime has a finite cost, while the parent captures the worst-case or long-run aggregate of these costs over an infinite execution.

Overview. The rest of the paper makes the flattening-based view explicit. Section 2 recalls NQAs and their decision problems, Sect. 3 describes QuAK's implementation of the flattening pipeline, and Sect. 4 evaluates the tool.

Related Work. NQAs were introduced in [11], which established their expressiveness beyond QAs and much of the decidability landscape. Bounded-width NQAs [9] restrict the number of concurrently active children, yielding better complexity in several cases. Quantitative monitor automata [10] provide a counter-based formulation expressively equivalent to bounded-width NQAs. QuAK [6,7] is the first tool to automate the analysis of QAs; we build directly on its codebase. Weighted automata tools such as Vaucanson [21], Vcsn [12], and Awali [20] primarily target finite words with semiring-style semantics. Signal temporal logic tools (e.g., Breach [13], S-TaLiRo [1], and RTAMT [22]), timed-automata and hybrid-systems tools (e.g., UPPAAL [19] and HyTech [16]), and probabilistic model checkers (e.g., PRISM [18] and Storm [15]) address orthogonal quantitative verification problems.

2 Theoretical Background

Quantitative Automata and Value Functions. A *quantitative automaton* (QA) over (finite or infinite) words is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F, C)$ consisting of a finite alphabet Σ, a finite state set Q, an initial state q_0, a transition relation $\delta \subseteq Q \times \Sigma \times Q$, an accepting-state set $F \subseteq Q$, and a weight function $C : \delta \to \mathbb{Q}$.

A *run* of $\mathcal{A}$ on a (finite or infinite) word $w = w_1 w_2 \dots$ is a sequence of states $\pi = q_0 q_1 \dots$ with $(q_{i-1}, w_i, q_i) \in \delta$ for all $i \geq 1$. Each run induces a weight sequence $C(\pi) = C(q_0, w_1, q_1), C(q_1, w_2, q_2), \dots$ A *value function* aggregates the weight sequence of a run into a single value. For infinite sequences, we consider:

$$\mathsf{Inf}(x) = \inf_{i\geq 1} x_i \qquad \mathsf{Sup}(x) = \sup_{i\geq 1} x_i$$
$$\mathsf{LimInf}(x) = \liminf_{i\to\infty} x_i \qquad \mathsf{LimSup}(x) = \limsup_{i\to\infty} x_i$$
$$\mathsf{LimInfAvg}(x) = \liminf_{k\to\infty} \tfrac{1}{k}\sum_{i=1}^{k} x_i \qquad \mathsf{LimSupAvg}(x) = \limsup_{k\to\infty} \tfrac{1}{k}\sum_{i=1}^{k} x_i$$

For finite sequences $x = x_1 \dots x_n$, we consider $\mathsf{Min}(x) = \min_i x_i$, $\mathsf{Max}(x) = \max_i x_i$, $\mathsf{Sum}^+(x) = \sum_{i=1}^{n} |x_i|$, $\mathsf{Sum}^-(x) = -\sum_{i=1}^{n} |x_i|$, and $\mathsf{Sum}^B(x)$, which equals $\sum_{i=1}^{n} x_i$ if all partial sums remain in $[-B, B]$ for a fixed bound B, and the first bound crossed ($-B$ or B) otherwise. For a QA $\mathcal{A}$ on infinite (resp. finite) words with value function f, a run π is *accepting* if it visits F infinitely often (resp. if it ends at a state in F), and the *value* of $\mathcal{A}$ on a word w is the supremum of $f(C(\pi))$ over all accepting runs π on w. An automaton is *deterministic* if each state has at most one outgoing transition per letter.

Nested Quantitative Automata. A *nested quantitative automaton* (NQA) [11] $\mathbb{A} = (\mathcal{A}, \mathcal{B}_1, \dots, \mathcal{B}_k)$ consists of a *parent automaton* $\mathcal{A} = (\Sigma, Q, q_0, \delta, F, L)$, where $(\Sigma, Q, q_0, \delta, F)$ is an automaton over infinite words and $L : \delta \to \{\bot, 1, \dots, k\}$ is a labeling function, and k *child automata* $\mathcal{B}_1, \dots, \mathcal{B}_k$, each a QA over finite words. A parent transition t with $L(t) = j \in \{1, \dots, k\}$ *invokes* (or *spawns*) child $\mathcal{B}_j$; a transition with $L(t) = \bot$ is *silent* and invokes no child. An NQA is an (f, g)-*automaton* if the parent uses the value function f and every child uses g.

A *run* of $\mathbb{A}$ on $w \in \Sigma^\omega$ is a tuple $(\Pi, \pi_1, \pi_2, \dots)$ where $\Pi = \Pi[0]\Pi[1]\dots$ is a run of the parent on $w = w_1 w_2 \dots$, and for each $i \geq 1$: if the i-th parent transition has label $j \in \{1, \dots, k\}$, then π_i is a finite run of $\mathcal{B}_j$ on the finite infix $w_i w_{i+1} \cdots w_{i'}$ of the same input word, for some endpoint $i' \geq i$; otherwise π_i is undefined. Here the i-th parent transition reads w_i, so the child spawned by that transition starts on the same input letter. When π_i is defined, the child invoked at position i terminates at position i', returning the value $g(C(\pi_i))$. The endpoint i' is part of the nondeterministic choice of the NQA run: a child spawned at position i may terminate at any position $i' \geq i$ at which it has an accepting finite run on $w_i \cdots w_{i'}$. The run $(\Pi, \pi_1, \pi_2, \dots)$ is *accepting* if: (i) the parent run Π visits F infinitely often, (ii) every invoked child run is finite and accepting, and (iii) infinitely many parent transitions are non-silent. In particular, every spawned child must be assigned a finite run ending in an accepting state; a parent run that leaves some child running forever is not accepting. The *value* of an accepting run is f applied to the sequence of returned child values, omitting silent transitions. The value of $\mathbb{A}$ on w is the supremum over all accepting runs on w. An NQA is *deterministic* if the parent and all children are deterministic, and accepting states in each child have no outgoing transitions. Thus, once a deterministic child reaches an accepting state, its termination position is forced; it terminates at the first accepting position reached by its unique run. Figure 1 shows a deterministic $(\mathsf{LimSupAvg}, \mathsf{Sum}^+)$-automaton computing average response time.

Decision Problems. Given a QA or NQA and a threshold $\lambda \in \mathbb{Q}$, the *emptiness* problem asks whether there exists $w \in \Sigma^\omega$ with value at least λ, and the

Table 1. Complexity of decision problems for NQAs with parent aggregator f and child aggregator g [11]. †The $(\mathsf{LimInfAvg}, \mathsf{Sum}^+)$ case is open; the others are in ExpSpace.

	$f \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$		$f \in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$	
	Emptiness	Universality	Emptiness	Universality
$g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^B\}$	PSpace	ExpSpace	PSpace	Undecidable
$g \in \{\mathsf{Sum}^+, \mathsf{Sum}^-\}$	PSpace	ExpSpace	ExpSp./Open†	Undecidable

universality whether every $w \in \Sigma^\omega$ has value at least λ. Table 1 summarizes the complexity landscape from [11]. QuAK supports all combinations that are known to be decidable.

Algorithmic Overview. The decision procedures reduce NQA threshold problems to QA threshold problems via *flattening*: constructing a QA that simulates the relevant information about active child states. We give the key ideas below; see [11] for the details.

(1) Regular children. For children with $g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^B\}$, the set of possible return values is finite, so for each value v, the words of value v form a regular language. This enables a reduction from (f, g)-automata to *semantically equivalent* f-automata with silent transitions [11, Lem. 4.10]: for each child invocation guess the child's return value, verify the guess via a DFA recognizing the corresponding language, and emit the guessed value as the transition weight. The construction incurs exponential blowup in the total size of these DFAs.

(2) Threshold-based bounding. For NQAs with $f \in \{\mathsf{Sup}, \mathsf{LimSup}, \mathsf{Inf}, \mathsf{LimInf}\}$ and $g \in \{\mathsf{Sum}^+, \mathsf{Sum}^-\}$, child return values exceeding a given threshold λ can be truncated without affecting the emptiness or universality decision [11, Thm. 4.18], reducing to the Sum^B case.

(3) Extremal parents with monotonic children. For emptiness checking of NQAs with $f \in \{\mathsf{Sup}, \mathsf{LimSup}, \mathsf{Inf}, \mathsf{LimInf}\}$ and $g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^+, \mathsf{Sum}^-\}$, we provide specialized guess-and-verify procedures that exploit two properties to achieve smaller state spaces than the general reduction. First, the child functions are *monotonic*: for Min and Max, we track the closest value to a guessed return value seen so far; for Sum^+ and Sum^-, the guessed value acts as a budget to produce or consume. This enables pruning configurations where children can no longer achieve their guessed return values. Second, the parent functions are *extremal*: they select a single weight rather than computing an aggregate like an average. The parent objective determines how many guessed values to track. For Inf and LimInf, the flattened automaton pairs each child state with a guessed return value, tracking all active children; states with dominated guesses are pruned. For Sup and LimSup, exceeding the threshold requires witnessing only one child return value, so we track a guessed value for one distinguished child at a time while other children are tracked only for termination. This yields substantially smaller state spaces for Sup and LimSup.

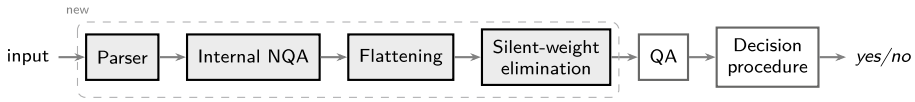

Fig. 2. The extended architecture of QuAK. Shaded components are new in this version. A nested quantitative automaton (NQA) is flattened into a standard quantitative automaton (QA) and fed to existing decision procedures.

(4) Limit-average with unbounded children. Recall that universality for limit-average QAs is undecidable, and this extends to NQAs with any child aggregator; we therefore focus on emptiness. For NQAs with $f \in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$ and $g = \mathsf{Sum}^{-}$, the construction proceeds in three phases [11, Thm. 4.20]. First, the automaton is determinized via alphabet extension, encoding each combination of nondeterministic choices into a new input letter. Second, child steps are synchronized so that all active children advance simultaneously. Third, a powerset-based flattening tracks bounded-multiplicity configurations. For emptiness of $(\mathsf{LimSupAvg}, \mathsf{Sum}^{+})$-automata, we first check whether child return values can grow unboundedly; if so, emptiness holds trivially, otherwise the problem reduces to the Sum^{B} case [11, Lem. 5.10]. The case of $(\mathsf{LimInfAvg}, \mathsf{Sum}^{+})$-automata remains open; unlike $\mathsf{LimSupAvg}$, the unboundedness check does not apply since unbounded child returns do not resolve emptiness for infimum-based objectives.

(5) Silent-weight elimination. The procedures above produce QAs with silent weights, arising from silent parent transitions, which invoke no child and therefore emit no returned value. These must be eliminated to obtain standard QAs. For extremal objectives, silent weights are replaced by neutral weights (e.g., $-\infty$ for Sup and LimSup, $+\infty$ for Inf and LimInf) that do not affect the computed value [11, Lem. 4.6]. For limit-average objectives we can eliminate silent weights via the path-compression construction of [11, Lem. 4.7], which replaces each maximal silent-weight segment by a single shortcut transition, possibly increasing the number of transitions.

(6) Antichain-based universality. We decide universality for NQAs by an on-the-fly search for a counterexample word, pruning the exploration using an antichain as in Forklift [14]. For the quantitative setting, contexts are enriched with value summaries: in addition to the reachable state set, we store for each state the current maximal weight bound, and extend the subsumption relation to compare both reachability and these bounds [6]. To incorporate Büchi acceptance in this version of QuAK, each context keeps two reachability summaries: all reachable pairs, and those reachable via a path that already visited an accepting state. We only evaluate candidate cycles against the weight threshold once acceptance progress has been witnessed.

3 QuAK's Architecture and Implementation

Overview. QuAK [6,7] is a C++ library and command-line tool for analyzing QAs. We extend QuAK to support NQAs. The key addition is a suite of *flattening*

procedures that compile an NQA into a QA with silent weights. This approach supports the known decidable emptiness and universality checks of all combinations of parent objectives in {Sup, Inf, LimSup, LimInf, LimSupAvg, LimInfAvg} with child objectives in {Min, Max, Sum^B, Sum^+, Sum^-}. Figure 2 depicts the pipeline; the rest of this section describes each phase and its implementation.

Input Format and Internal Representation. QuAK reads automata from plain text files as transition lists `a : v, q -> p` (letter `a`, weight `v`, from state `q` to `p`), with the initial state being the source of the first transition. We extend this format with an optional `final` directive for Büchi acceptance; otherwise, all states are accepting. Internally, a QA is stored as an `Automaton` object that does not fix a value function, allowing the same structure to be analyzed under different objectives. Automaton states are integer-indexed, and transitions are stored in adjacency-list form with both outgoing and incoming edges. While constructing an `Automaton` object, QuAK prunes unreachable states and caches its SCCs [6]. QuAK requires automata to be *complete*: every state must have at least one outgoing transition per letter. Determinism is not required, but QuAK detects it and dispatches to specialized routines when available.

Input NQAs consist of one `@PARENT` block and one `@CHILD` i block per child. Children use the syntax above, while parent transitions use `a : i, q -> p`, where $i > 0$ invokes child i and $i = 0$ denotes a silent transition. The parent uses Büchi acceptance, and children use finite-word acceptance. Internally, the `NestedAutomaton` and `ChildAutomaton` classes extend QuAK's base `Automaton` class. A `NestedAutomaton` object has a parent automaton of type `Automaton` and a vector of pointers to `ChildAutomaton` objects.

Flattening Nested Quantitative Automata. The flattening module implements the constructions; see Sect. 2 for the algorithmic details.

(1) Regular children. For NQAs with $g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^B\}$, QuAK implements an obligation-based flattening that preserves semantic equivalence. When a parent transition invokes a child, the flattened automaton guesses the child's return value on that transition; the guess is verified by an obligation tracking possible child configurations in parallel. Each active obligation records the child index, the guessed value, and a frontier of (*childState*, *accumulatedValue*) pairs; configurations that cannot reach a final state with the guessed value are pruned via target-aware reachability tables from reverse BFS. Return-value sets are computed parent-aware by exploring the synchronized parent–child product restricted to parent states that can still reach an accepting SCC. QuAK applies this procedure to universality for $f \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$ and $g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^B\}$, and to emptiness when $g = \mathsf{Sum}^B$ or when $f \in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$ and $g \in \{\mathsf{Min}, \mathsf{Max}\}$; remaining emptiness cases use specialized procedures.

(2) Threshold-based bounding. For universality of NQAs with $f \in \{\mathsf{Sup}, \mathsf{LimSup}, \mathsf{Inf}, \mathsf{LimInf}\}$ and $g \in \{\mathsf{Sum}^+, \mathsf{Sum}^-\}$, QuAK reduces the unbounded child aggregator to Sum^B via threshold-preserving clipping: values at or above the thresh-

old collapse to it for Sum^+; values below collapse to a strictly smaller value for Sum^-. The resulting bounded-return automaton is handled by the procedure above. The same reduction applies semantically to the corresponding emptiness cases, but QuAK uses the specialized threshold constructions below.

(3) Extremal parents with monotonic children. For emptiness with $f \in \{\mathsf{Sup}, \mathsf{LimSup}, \mathsf{Inf}, \mathsf{LimInf}\}$ and $g \in \{\mathsf{Min}, \mathsf{Max}, \mathsf{Sum}^+, \mathsf{Sum}^-\}$, QuAK avoids computing exact child return values. Instead, each child invocation is reduced to a binary threshold outcome: whether the child can return a value at least λ. For Min and Max, the verifier only needs to remember the current below/above-threshold status of each tracked child run; for Sum^+ and Sum^-, it additionally stores capped threshold progress. The implementation precomputes backward liveness information from final child states and uses it to prune guesses that can no longer realize their threshold outcome. The flattened automaton therefore has weights in $\{0, 1\}$, and emptiness is checked with the same parent aggregator f against threshold 1. For $f \in \{\mathsf{Sup}, \mathsf{LimSup}\}$, QuAK uses a single-witness variant: the flattened state stores one explicitly tracked threshold-reaching child, while the remaining active children are represented only by termination obligations.

(4) Limit-average with unbounded children. For $(\mathsf{LimSupAvg}, \mathsf{Sum}^+)$ emptiness, QuAK separates the unbounded case from the bounded one. It computes a threshold λ capturing the largest value a single child can accumulate without forcing a repetition of some global configuration with a strictly larger accumulated sum, and checks whether the instance is nonempty against λ. If this check succeeds, the instance is nonempty; otherwise the relevant child values are bounded, and QuAK calls the regular Sum^B flattening with $B = \lambda$. For $(\mathsf{LimInfAvg}, \mathsf{Sum}^-)$ and $(\mathsf{LimSupAvg}, \mathsf{Sum}^-)$ emptiness, QuAK first prepares the input for the synchronization construction, completing or determinizing via alphabet extension, it when needed. It then replaces all children by one synchronized ultimate child and flattens by tracking a bounded multiset of active ultimate-child states. The multiset is stored sparsely as sorted (*stateId*, *count*) pairs; BFS generates states on demand and drops successors whose multiplicity exceeds the computed bound c.

(5) Silent-weight elimination. Before passing the flattened automaton to the existing decision procedures, QuAK eliminates silent-weight transitions. For prefix-independent objectives (LimSup, LimInf, $\mathsf{LimSupAvg}$, $\mathsf{LimInfAvg}$), QuAK reduces the potential overhead by only considering those within accepting SCCs. This optimization is enabled by default, but can be deactivated.

(6) Antichains, interfacing, and optimizations. The flattening routines are used internally by the nested emptiness and universality checks, and exposed through the public API for custom analysis pipelines. For limit-average emptiness of QAs with Büchi acceptance, QuAK computes maximum mean cycles separately within accepting SCCs, ignoring non-accepting ones. For FORKLIFT inclusion [14], contexts now store two relations per weight level: all reachable target pairs, and the subset reachable via a path through an accepting state. The lasso membership test was likewise adjusted to require an accepting visit on the target

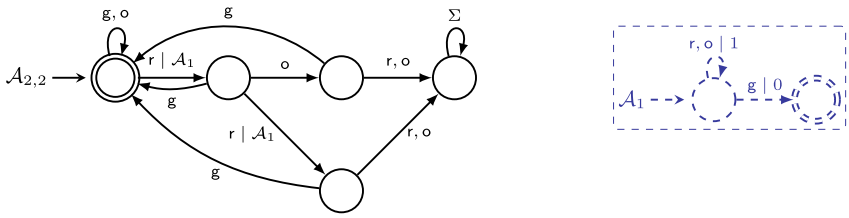

Fig. 3. The response-time automaton $\mathbb{A}_{2,2} = (\mathcal{A}_{2,2}, \mathcal{A}_1)$. Every parent transition without a child automaton annotation is silent.

cycle, ensuring inclusion counterexamples respect Büchi acceptance rather than only the weight threshold.

Availability and Usage. QuAK is open-source under the MIT license and available online.[1] The tool has no external dependencies other than a C++17 compiler. Instructions and examples are provided in the repository's README.

4 Experimental Evaluation

We evaluate QuAK's new capabilities along two benchmarks: (i) *response-time* properties and (ii) *resource-consumption* constraints. The experiments are designed to isolate the main sources of blowup in the flattening constructions: child return-value range, number of simultaneously active children, child-state space, and transition density of the flattened automaton.

The response-time benchmark is based on the standard example from [11], with parameters added to control the number of simultaneously pending requests and the response-time bound. The resource-consumption benchmark follows the construction of [11, Thm. 6.2], where it is used to show an exponential succinctness gap between deterministic NQAs and nondeterministic QAs. The generator scripts are included with the artifact.

Setup. All experiments were run on Ubuntu 24.04.3 with an Intel Ultra 7 Processor 255U and 32 GB RAM. The tool was compiled with GCC using C++17 and optimization level `-O3`. Each instance was limited to 300 s and 30 GB of memory. Reported runtimes are wall-clock seconds averaged over three runs, excluding parsing time; each measurement was preceded by a warm-up run.

4.1 Response Time

We study a parametric family of NQAs $\mathbb{A}_{n,k}$ for bounded-response properties in request-grant protocols. The parameter n bounds the number of simultaneously pending requests, and k bounds the allowable response time. The alphabet

[1] https://github.com/ista-vamos/nested-quak.

Table 2. Response time benchmarks: runtime in seconds, m = out of memory (30 GB), t = out of time (300 s).

(a) (Sup,Sum$^+$) emptiness

$n\backslash k$	4	8	16	32	64	128	256	512
4	.00	.00	.01	.01	.06	.32	1.5	6.3
8	–	.00	.01	.03	.16	.76	3.3	14
16	–	–	.01	.07	.39	1.7	7.3	32
32	–	–	–	.13	.85	3.8	17	72
64	–	–	–	–	1.3	7.8	35	m
128	–	–	–	–	–	12	71	m
256	–	–	–	–	–	–	m	m
512	–	–	–	–	–	–	–	m

(b) (LimSupAvg,Sum$^+$) emptiness

$n\backslash k$	3	4	5	6	7	8	9	10
3	.00	.00	.00	.01	.02	.05	.13	.31
4	–	.00	.00	.01	.07	.28	1.1	3.2
5	–	–	.00	.03	.21	1.3	5.5	20
6	–	–	–	.03	.29	2.8	18	93
7	–	–	–	–	.33	4.3	39	284
8	–	–	–	–	–	4.7	58	m
9	–	–	–	–	–	–	63	m
10	–	–	–	–	–	–	–	m

(c) (Sup,Sum^B) universality

$n\backslash k$	2	3	4	5	6	7	8	9
2	.00	.00	.01	.04	.14	.42	1.2	3.6
3	–	.00	.02	.08	.40	2.3	12	44
4	–	–	.02	.12	1.1	12	159	t
5	–	–	–	.14	2.0	51	t	t
6	–	–	–	–	2.4	136	t	m
7	–	–	–	–	–	177	t	m
8	–	–	–	–	–	–	t	m
9	–	–	–	–	–	–	–	m

comprises a request r, a grant g, and a neutral action o. For $n \geq 1$ and $k \geq n$, the parent automaton has $\frac{n(2k-n+1)}{2} + 2$ states and uses a single two-state child automaton. A child instance is spawned on each request, so at most n instances are concurrently active; if this bound is exceeded or some request goes ungranted for more than k steps, the parent moves to a rejecting sink. Each child accumulates the number of steps between its spawning request and the matching grant, and the parent value function determines how these per-request response times are combined: Sup yields the worst-case response time, while LimSupAvg yields the long-run average. Figure 3 shows a representative instance $\mathbb{A}_{2,2}$.

Unlike the unbounded response-time automaton in Fig. 1, the bounds enforced by $\mathbb{A}_{n,k}$ yield a finite-state benchmark suitable for systematic experiments. This family also captures a standard verification scenario: given a finite-state model of a server, quantify the worst-case or average response time over all permitted behaviors. We evaluate QuAK on this benchmark with emptiness of (Sup, Sum$^+$)-automata, emptiness of (LimSupAvg, Sum$^+$)-automata, and universality of (Sup, Sum^B)-automata. The results are reported in Table 2c.

Emptiness of (Sup, Sum$^+$)-automata Table 2a shows that emptiness checking scales well on this family: the dependence on k is close to quadratic, while the dependence on n is comparatively mild. This behavior is consistent with the specialized procedure for extremal parents with monotonic children (Sect. 2) and with the structure of $\mathbb{A}_{n,k}$. All children are copies of the same deterministic two-state automaton: before the next grant, every active child is in the same nonfinal state and differs only by accumulated Sum$^+$ value. The specialized construction stores capped threshold progress only for one distinguished witness; the other active children contribute identical termination obligations, so increasing n mainly enlarges the parent queue state space.

Emptiness of (LimSupAvg, Sum$^+$)-Automata Table 2b shows a different pattern: runtime grows quickly with both parameters, with the feasible range of n much smaller than in Table 2a. For these bounded-response instances, the unbounded-child case is ruled out and QuAK falls back to regular Sum^B flattening. Each spawned child is represented by a guessed response time and an obligation verifying the guess until the next grant. There are $O(k)$ possible guesses per child, and up to n such obligations may be live simultaneously. Thus k enlarges

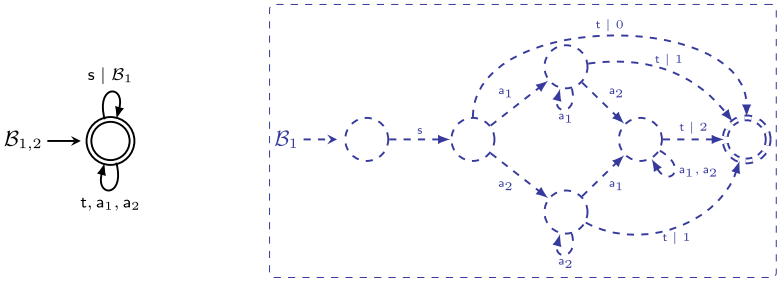

Fig. 4. The resource-consumption automaton $\mathbb{B}_{1,2} = (\mathcal{B}_{1,2}, \mathcal{B}_1)$. Every parent transition without a child automaton annotation is silent. The rejecting sink state of the child automaton and transitions to it are not shown. Every child transition without a weight annotation has weight 0.

the per-child value domain, while n determines the combinatorial search space by increasing the number of active guesses that must be combined.

Universality of (Sup, Sum^B)-Automata Table 2c shows that universality follows a similar pattern: runtime grows rapidly in both parameters and reaches time or memory limits at smaller values than in the emptiness experiments. The source of growth is the same: regular Sum^B flattening combines $O(k)$ response-time guesses across up to n concurrently active children. Universality is consistently more expensive because handling the flattened automaton's nondeterminism requires a PSPACE procedure rather than the PTIME emptiness check.

4.2 Resource Consumption

We study another parametric family of NQAs $\mathbb{B}_{n,k}$ that monitors resource consumption in a system with dynamically started and terminated processes. The parameter n bounds the number of concurrently running processes, and k bounds the number of distinct resources. The alphabet comprises $n(k+2)$ actions: for each process $i \in \{1, \ldots, n\}$, a start action s_i, a termination action t_i, and k resource-access actions $a_{i,1}, \ldots, a_{i,k}$. For $n \geq 1$ and $k \geq 1$, the parent automaton has a single control state and uses n child templates $B_1, \ldots, B_n$, each with $2^k + 3$ states. When process i starts, the parent spawns an instance of child B_i, which remains active until the matching termination t_i. While active, the instance tracks which resources process i accesses; upon termination, it returns the number of distinct resources used, computed via the Max value function. The child rejects the words that violate the protocol, such as starting an already-running process. The parent aggregates per-process values with either Sup, yielding the maximal consumption across executions, or LimSupAvg, yielding the long-run average. Figure 4 shows a representative instance $\mathbb{B}_{1,2}$.

This family provides a controlled benchmark with two orthogonal scaling axes: increasing k multiplies the resource-usage patterns a process can exhibit,

Table 3. Resource consumption benchmarks: runtime in seconds, `m` = out of memory (30 GB), `t` = out of time (300 s).

(a) (Sup,Max) emptiness

$n\backslash k$	1	2	3	4	5	6
1	.00	.00	.00	.00	.00	.00
2	.00	.00	.01	.06	.37	1.9
3	.01	.10	1.1	11	103	m
4	.12	2.4	43	m	m	m
5	1.3	38	m	m	m	m
6	9.9	m	m	m	m	m

(b) (LimSupAvg,Max) emptiness

$n\backslash k$	1	2	3	4
1	.00	.00	.00	.00
2	.00	.03	1.1	62
3	.12	48	t	t
4	20	t	t	m

while increasing n multiplies concurrent process overlap. Although $\mathbb{B}_{n,k}$ is deterministic as a nested automaton, any equivalent non-nested automaton must encode joint resource subsets explicitly and can therefore grow exponentially in the number of concurrently tracked processes [11]. We evaluate QuAK on emptiness of $(\mathsf{Sup}, \mathsf{Max})$- and $(\mathsf{LimSupAvg}, \mathsf{Max})$-automata; results appear in Table 3.

Emptiness of (Sup, Max)-Automata Table 3a shows rapid blowup in both parameters. Each child encodes which resource subset a process has accessed, giving $\Theta(2^k)$ control states per process. The Sup construction tracks one witness (whether some process sees all resources) but still carries termination obligations for the remaining children. Since a run may have one active instance per process template, these obligations yield a $2^{\Theta(nk)}$ joint space.

Emptiness of (LimSupAvg, Max)-Automata As Table 3b shows, the limit-average case becomes substantially harder than the supremum once multiple processes overlap. The Sup procedure needs only one threshold-reaching witness, whereas the LimSupAvg regular-children procedure must preserve the exact return values of all active children. These returns range over $0, \ldots, k$, and each verifier still carries its child's resource-subset state. Combined with the $2^{\Theta(nk)}$ subset space, this exceeds resource budgets at small parameter values.

5 Conclusion

We presented the first tool for analyzing nested quantitative automata (NQAs), extending QuAK with flattening procedures that reduce nested emptiness and universality to their non-nested counterparts. This enables automated verification of quantitative properties beyond standard quantitative automata, such as average response time. Our experiments show that explicit flattening can incur large state spaces, especially when many child instances are active. Symbolic state representations [5] and domain-specific reductions [4] could mitigate this blowup. Identifying tractable fragments, informed for example by safety-liveness classifications [2,3,17], would broaden the tool's applicability.

Acknowledgments. This work was supported by the European Research Council (ERC) Grants VAMOS (No. 101020093) and HYPER (No. 101055412).

Data Availability Statement. The artifact supporting the experimental results in this paper is available in the QuAK repository at https://github.com/ista-vamos/nested-quak. It contains the extended QuAK implementation, benchmark generators, example inputs, and scripts/logs for reproducing the reported tables. The artifact is intended to reproduce the experiments under the setup described in Sect. 4; runtimes may vary across machines, and the reported timeout and memory-exhaustion results depend on the stated hardware limits. No sensitive or restricted data are used. An archived version is available on Zenodo at DOI: http://doi.org/10.5281/zenodo.19844606.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. 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.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
2. 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, 18–23 September 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
3. Boker, U., Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Safety and liveness of quantitative properties and automata. Log. Methods Comput. Sci. **21**(2) (2025). https://doi.org/10.46298/LMCS-21(2:2)2025
4. Bokor, P., Kinder, J., Serafini, M., Suri, N.: Supporting domain-specific state space reductions through local partial-order reduction. In: 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, pp. 113–122. IEEE (2011). https://doi.org/10.1109/ASE.2011.6100044
5. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. 98(2), 142–170 (1992). https://doi.org/10.1016/0890-5401(92)90017-A
6. Chalupa, M., Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Quak: quantitative automata kit. In: Margaria, T., Steffen, B. (eds.) ISoLA 2024, Part IV. LNCS, vol. 15222, pp. 3–20. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-75387-9_1
7. Chalupa, M., Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Automating the analysis of quantitative automata with quak. In: Gurfinkel, A., Heule, M. (eds.) TACAS 2025, Part I. LNCS, vol. 15696, pp. 303–312. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-90643-5_16
8. 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
9. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted limit-average automata of bounded width. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) 41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, Kraków, Poland, 22–26 August 2016. LIPIcs, vol. 58, pp. 24:1–24:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPICS.MFCS.2016.24

10. Chatterjee, K., Henzinger, T.A., Otop, J.: Quantitative monitor automata. In: Rival, X. (ed.) SAS 2016. LNCS, vol. 9837, pp. 23–38. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53413-7_2
11. Chatterjee, K., Henzinger, T.A., Otop, J.: Nested weighted automata. ACM Trans. Comput. Log. **18**(4), 31:1–31:44 (2017). https://doi.org/10.1145/3152769
12. Demaille, A., Duret-Lutz, A., Lombardy, S., Sakarovitch, J.: Implementation concepts in Vaucanson 2. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 122–133. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39274-0_12
13. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
14. Doveri, K., Ganty, P., Mazzocchi, N.: Forq-based language inclusion formal testing. In: Shoham, S., Vizel, Y. (eds.) CAV 2022, Part II. LNCS, vol. 13372, pp. 109–129. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13188-2_6
15. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
16. Henzinger, T.A., Ho, P.-H.: HyTech: the Cornell hybrid technology tool. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1994. LNCS, vol. 999, pp. 265–293. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60472-3_14
17. Henzinger, T.A., Mazzocchi, N., Saraç, N.E.: Quantitative safety and liveness. In: Kupferman, O., Sobocinski, P. (eds.) FoSSaCS 2023, ETAPS 2023. LNCS, vol. 13992, pp. 349–370. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30829-1_17
18. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46029-2_13
19. 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
20. Lombardy, S., Marsault, V., Sakarovitch, J.: Awali, a library for weighted automata and transducers (version 2.3) (2022). http://vaucanson-project.org/Awali/2.3/
21. Lombardy, S., Poss, R., Régis-Gianas, Y., Sakarovitch, J.: Introducing Vaucanson. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 96–107. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45089-0_10
22. Ničković, D., Yamaguchi, T.: RTAMT: online robustness monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 564–571. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_34