Advancing the Theory of Quantitative Algorithmic Monitoring

Research Proposal

N. Ege Saraç

IST Austria

Abstract

Runtime verification (RV) is a lightweight, dynamic technique where a monitor watches a trace of a system and, if possible, decides after observing each finite prefix whether or not the unknown infinite trace satisfies a given specification. Theoretically, RV moves the burden from emptiness checking in static verification to membership checking, an easier problem. This shift introduces the opportunity to use more powerful formalisms. We suggest using a machine model that enables reasoning about quantitative information and moving RV to a quantitative setting. Such a setting is attractive because quantitative verdicts can be approximate and thus compared regarding their precision. We plan to develop a framework for online and best-effort quantitative monitoring that subsumes (i) a cost-centric theory of monitorability and (ii) a precision-cost theory of approximate monitoring. Moreover, we aim to extend the framework to monitors that take corrective action and decentralized monitoring to improve our framework's practical relevance.

1 Introduction

The impact of software on society is increasing rapidly, along with our dependence on it. This dependence grows at such a speed that the need for reliable software is rising faster than ever. Research on static verification (SV) has provided valuable techniques such as model checking and theorem proving for ensuring before deployment whether a software system is well-behaved. However, these established methods face severe challenges as the complexity of software systems increases. Although SV tools are also improving, there is still a growing gap between their capabilities and the complexity of real systems.

Runtime verification (RV) is a lightweight, dynamic verification technique that evaluates the current system execution with respect to a property. The primary construct of RV is the *monitor*, which observes inputs and outputs of the system under scrutiny and issues a verdict in real-time after each observation.

While the increasing software complexity poses challenges for SV, the positive trend in hardware parallelism makes using dynamic techniques like RV more practical. Although many-core processors and computer clusters are abundant, it is hard to get the maximal performance out of such systems. Therefore, one can argue that not all resources will go into performance and some can be used for increased assurance by checking the properties of systems on the fly.

Instead of considering *all* possible system behaviors, RV asks whether a *single*, given behavior satisfies the specification. Theoretically, this is a shift from the *emptiness* problem to that of *membership*. Since the membership problem is easier to solve, it yields an opportunity for more expressive formalisms. In particular, we can reason about a variety of *quantitative* information, which is interesting for a reason beyond being non-boolean: it can be *approximate*. To capture this, we use monitors with integer-valued registers.

We advocate for the use of monitors as third-party components that run in parallel to the system and provide a best-effort assurance on its reliability and robustness. Therefore, keeping practicality in mind, we aim to provide a theoretical framework for monitoring quantitative properties with various degrees of precision and capture essential design trade-offs on the "quality" of monitors.

The rest of this proposal is organized as follows: In Section 2 we discuss the related work and identify the gap for a theory of infinite state monitors as well as quantitative and approximate monitoring. In Section 3 we present our main research goals and the assumptions under which we aim to tackle the problems. In Section 4 we describe our recent work that serve as a foundation for the research directions in line with our goals. In Section 5 we focus on future research directions that stem from our goals and progress.

2 Related Work

Specifying and synthesizing monitors. Synthesizing monitors automatically from given specifications is a significant research topic in RV. One of the most popular specification languages in RV is linear temporal logic (LTL). Such propositional approaches to specification and synthesis of monitors are studied extensively over the years [66, 71, 67, 72, 88].

An alternative, parameterized approach (where events carry data) recently gained popularity. Inspired by the use of rule-based production systems in artificial intelligence, Havelund [69] proposed rule-based monitoring. In this setting the system states are considered as a set of facts and the specifications are taken as "rules" of the form *conditions* $\Rightarrow$ *action*, which suits well to the processing of data languages. The monitoring algorithm here simply tries to match states with conditions efficiently.

Another work in this direction is by Chen and Roşu [43], where they introduced parametric trace slicing, which enables monitoring specifications where a behavior is associated with each set of data values. Monitoring in this setting consists of two parts. First, we slice each trace based on the data values associated with it, and then we check each slice to see whether it it satisfies the specification.

Another significant approach is stream-based RV. Inputs and outputs can be seen as streams of data in this context, which makes RV a special case of stream processing. The first stream-based framework for RV is Lola [46] which focuses on monitoring synchronous streams. It has a flexible design where users write stream expressions that define the relationship between inputs and outputs. Since it is not limited to the boolean values, Lola enables quantitative analysis of system behaviors. The monitoring algorithm incrementally computes the output values based on the stream expressions defined by the user and the previous values in the streams. This line of work is later extended by Finkbeiner et al. with template stream expressions and dynamic stream generation in Lola 2.0 [57] and for real-time monitoring with asynchronous input streams in RTLola [28]. Another stream-based framework is TeSSLa by Leucker et al. [45, 79, 80] which was initially designed for asynchronous streams.

Following these works, researchers suggested many methods for synthesizing monitors from a variety of specification languages. To count a few, extensions of temporal logic with freeze quantifiers [48], counters [51], first-order quantifiers [70] and time intervals [22]; extensions of finite automata with quantified event parameters [17], registers [68], and discrete clocks [33]. In the branching-time setting, a prominent line of work is by Aceto et al. [63, 3, 2, 4], which utilizes the μ-calculus and Hennessy-Milner logic.

We aim to study automata with integer-valued registers, which is a flexible and powerful model, to enable the detailed analysis trade-offs in monitor design. Although infinite-state monitors have been used in practice for years, their theoretical properties are not systematically studied before.

Monitorability of specifications. The notion of monitorability bridges the gap between what a specification means and whether it can be recognized during a system execution. The first definition of monitorability by Kim et al. [77] focused on detecting violations of a property. It corresponds to a strict subset of safety properties over infinite words because this definition of monitorability requires the set of finite prefixes of the property to be co-recursively enumerable while the notion of safety is not limited by computational considerations.

This rather restrictive definition was later generalized by Pnueli and Zaks [88]. The authors defined monitorability of a property relative to a finite trace considering both satisfactions and violations. According to their definition, a property is s-monitorable for a finite trace s if there is a finite continuation r such that either every infinite continuation to sr satisfies the property or every infinite continuation violates it.

Following the definition of Pnueli and Zaks, Bauer et al. [26] defined a property as monitorable if it is s-monitorable for every finite trace s. This definition is considered the classical definition of monitorability.

The authors showed that the set of monitorable properties under this definition strictly contains the union of safety and co-safety properties. This result was later improved by Falcone et al. [53] by showing that it strictly contains finite, positive boolean combinations of safety and co-safety properties. Moreover, Diekert and Leucker [49] characterized monitorable properties topologically as sets whose boundary is nowhere dense.

Building on the work of Bauer et al. [25], Falcone et al. [53] proposed a definition of monitorability with parameterized truth domains instead of using a fixed, three- or four-valued domain for monitoring. This enables a finer classification of monitorable properties based on the truth domain. The authors show how their definition of monitorability relates with the safety-progress hierarchy [37] considering several truth domains. Notably, according to their definition, every (linear-time) property is monitorable in a four-valued domain where the usual "inconclusive" verdict is split into "currently true" and "currently false" verdicts.

In addition to these definitions, a framework that captures monitorability in both linear- and branching-time was studied by Aceto et al. [2] and a process-algebraic approach to monitoring by Francalanza [62].

We aim to develop a cost-centric theory of monitorability through a machine model that allows reasoning about both boolean specifications with quantitative aspects and purely quantitative specifications. Such a fine-grained approach to monitorability is novel to the best of our knowledge.

Monitoring specifications with quantitative flavor. Quantitative specifications generalize boolean specifications by mapping infinite traces to a quantitative domain (e.g., integers) instead of true or false. Chatterjee et al. defined quantitative languages in [40] where they used weighted automata to study several classes of such languages and investigate classical decision problems such as emptiness, universality, inclusion, and equivalence. They also studied expressiveness and closure properties of quantitative languages [39]. To account for quantitative aspects of systems, an extension of LTL with discounting is studied in [7], with averaging in [34], computation tree logic and LTL with accumulation assertions in [31]. In addition, quantitative properties has received significant attention also in the probabilistic setting [78, 38, 60, 15, 35], and quantitative frameworks for comparing traces and implementations for the same boolean specification were studied in [36, 30].

Chatterjee at al. also defined and studied nested weighted automata [42] where a "master" automaton spawns "slave" automata for the computation of sub-components of a property and combines the values returned over finite prefixes by the slaves to approximate the property value of an infinite trace. Later, they defined and studied quantitative monitor automata [41] where they augment weighted ω-automata with monitor counters or weighted automata themselves. They show that the two augmentations yield an equivalent expressive power when the nested formalism has a bound on the number of slaves. An equally expressive logic was proposed in [84]. Despite its potential for a use in RV, these works mainly focused on decidability and rather coarse expressiveness issues.

Another significant line of work in this direction is by Alur et al. In [9], they defined cost-register automata which serve as an effective specification language that enables the analysis of "regular" functions from finite traces to cost domains. Moreover, in [11], they defined quantitative regular expressions which is an alternative abstraction to cost-register automata with an equivalent expressive power. They built on their previous work by proposing a modular framework based on weighted automata augmented with nesting and parallelism for processing streams of data in [12] and a model called data transducer which has the same expressive power as cost-register automata while being exponentially more succinct in [13]. Most recently, they developed a quantitative stream processing theory in [10] using cost-register automata where they also established that weighted automata is a special case of cost-register automata. Although all these works consider the quantitative aspects of system specifications and verification, they mainly focus on runtime decidability issues for boolean specifications over streams of data events, but they do not consider approximate monitoring at varying degrees of precision.

A relevant collection of literature on quantitative approaches in RV features primarily the signal temporal logic (STL) [83]. In [50], the authors study the notion of robustness for STL and a method for its efficient computation. Quantitative monitoring of STL by using edit distance between behaviors is studied in [76], which also has a flavor of approximate monitoring due to the use of quantization of real-valued signals. A first-order logic for studying the specification of real-valued temporal behaviors is proposed in [16]. Several trade-offs regarding monitor quality also arise in this context, mostly due to the processing of continuous signals. For example, it is shown in [1] that sampling rate and the quality of robustness computation is tightly connected.

We aim to develop a theory for monitoring generic quantitative properties with a focus on precision-cost trade-offs. A rigorous study on this direction has not been carried out before.

Use of monitors for real systems. Passive monitors designed for systems with a single computing unit have limited practical relevance. Especially in an online setting, active monitoring can be helpful by steering the system away from undesired situations. Also, due to the abundance of distributed systems, it is also necessary to ensure that our framework can handle such systems effectively.

The branch of RV focusing on active monitors is called runtime enforcement (RE). In RE, the main goal is to actively ensure that the system conforms to the desired property during runtime, while complying with the two important constraints: soundness (modified behaviors must be correct) and transparency (correct behaviors must not be modified). The notion of enforceability connects what a specification means and whether it can be imposed during a system execution. Schneider proposed the model of security automata for enforcement [90]. This model essentially halts the system execution whenever it recognizes a bad prefix, which means that the set of enforceable properties are limited to safety properties. Later, the model of edit automata was proposed and studied by Ligatti et al. [27]. In addition to halting on bad prefixes, this model can insert and suppress events. Notably, these abilities enable enforcing also some liveness properties [81, 82]. Falcone et al. defined and studied generalized enforcement monitors [56] which build on the previous models. They also showed that the set of enforceable properties are exactly the response properties [53] of the safety-progress hierarchy [37]. In addition, models of enforcement with memory constraints were studied in [59, 92, 29], for timed properties in [89, 20, 85], with predictive semantics in [87, 86], in the branching-time setting in [5, 6].

Decentralized monitoring is studied first by Bauer and Falcone [24]. The authors developed a rewriting method for automatically distributing LTL specifications under the assumption of a global clock, which was later generalized by Falcone et al. to regular specifications [52]. The main advantage of this method is the lack of a central monitor, and thus less communication overhead. Similar approaches in this direction is further studied for real-time systems [18], with a hierarchical approach to communication in [44], with a stream-based approach in [47], with an online approach using low-resolution global clocks in [19]. The asynchronous setting for safety monitoring was studied in [91], a process-algebraic method in [64], an adaptive approach for spatial specifications in [14]. Monitoring of distributed systems where parts of the system can fail was considered in [21, 23, 32, 61]. Decomposition of monitors and specifications was investigated in [54, 65].

We aim to carry the theories of RE and distributed monitoring to the quantitative setting while enabling the consideration of precision-cost trade-offs and approximations for specifications with quantitative aspects, both of which are novel in RV.

3 Objectives and Assumptions

We set two main research goals based on our observations presented in the previous sections.

Cost of monitorability. Since we can move beyond formalisms with decidable emptiness to those with decidable membership, we model our monitors as automata with integer-valued registers that operate in real-time (i.e., bounded number of monitor operations per system operation) and verdicts in a quantitative domain. In this setting, we aim to explore the "cost" of monitorability systematically. One dimension of cost is the *resource use*, e.g., number of registers or types of operations a monitor can carry out. Another is *time*, e.g., number of operations per observation or number of observations needed to reach a definite verdict. This exploration also includes the cost-bounded monitor synthesis problem: given a property and a cost limit, how can we construct a monitor for the given property within the limit? Two related directions onto which the cost-centric theory should extend are (i) quantitative enforcement monitors that can take corrective actions on the system, and (ii) decentralization of quantitative monitors and specifications.

Precision of approximate monitoring. Quantitative specifications come in different forms, e.g., purely quantitative as in maximal response time, or in combination with boolean specifications as in discounted safety. Despite the different forms, there is always a natural basis for *approximate* monitoring. We aim to build a framework for monitoring that captures the notions of *precision* and *robustness* in a generic quantitative setting. As these notions raise many questions regarding monitor resources, the framework must enable the analysis

of precision-cost trade-offs, which we take as a central design criterion for monitors. Moreover, it should be sufficiently flexible to extend easily to synthesis, enforcement, and decomposition problems. A meticulous study on this front is needed to facilitate the practicality of quantitative monitoring.

Monitors can be used in many different ways – see [55] for a detailed classification of monitoring settings. In line with our goals, we make the following assumptions:

1. System components are black-boxes.
2. Monitors are deterministic and operate in real-time.
3. Monitors are isolated from system components.
4. Monitors must be as non-intrusive as possible.

4 Progress

In this section we describe our recent work in algorithmic monitoring and briefly discuss our contributions in line with the research goals presented in previous sections.

4.1 Safety monitoring with integer-valued registers [58]

While finite-state monitors have been studied extensively, in practice, monitoring software also makes use of unbounded memory. We define a model of automata equipped with integer-valued registers which can execute only a bounded number of instructions between consecutive events, and thus can form the theoretical basis for the study of infinite-state monitors. We classify these monitors according to the number of available registers, and the type of register instructions.

A *register monitor* is equipped with a finite set of control locations, a finite set of registers, and an instruction set consisting of arithmetic operations according to which the registers are tested and updated. It operates over a finite alphabet and it is deterministic. A configuration of the monitor consists of a control location and a register valuation. A transition is a relation between configurations such that (i) the monitor moves from the previous control location to the next control location with the given input letter, (ii) the previous register valuation satisfies the test associated with the transition, and (iii) the next register valuation is in accordance with the previous valuation and the updates associated with the transition. A run is a valid sequence of transitions, and an infinite trace is accepted by the monitor if it has a run. The language of the monitor is the set of infinite traces accepted by the monitor. In Figure 1 below we give an example of a register monitor.

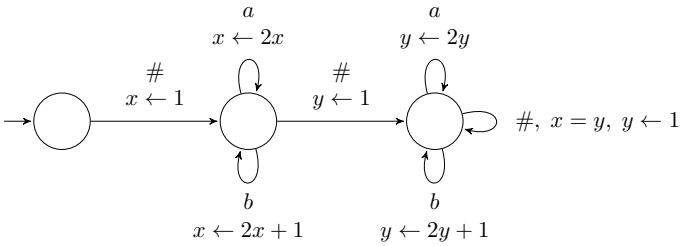

Figure 1: A $\langle 1, +1, +, = \rangle$-monitor recognizing the language $L = \bigcup_{w \in \Sigma^*} \#(w\#)^\omega \cup \#(w\#)^*\Sigma^\omega$. This language consists of infinite sequences starting with # in which one unique finite word over $\Sigma = \{a, b\}$ repeats, each consecutive pair of occurrences separated by #. The language L is recognized by the real-time adder monitor with 2 registers. The part of a word before a separator, if any, is encoded in x using a binary representation. Later occurrences of finite words are encoded in y. At every separator the encoding of the two words must match.

First, we focus on relatively simple $\langle +1, = \rangle$-monitors, called *counter monitors*. We show that for every $k \geq 1$ there is a real-time $(k + 1)$-counter monitor without an equivalent k-counter monitor. For this, we

consider the alphabet $\Sigma_k = \{0, 1, \ldots, k\}$ and the property $P_k = \{f \in \Sigma_k^\omega \mid \forall\, 0 \leq i < k : \forall s \prec f : |s|_i \geq |s|_{i+1}\}$ where $|s|_i$ denotes the number of occurrences of letter $i \in \Sigma_k$ in $s \in \Sigma_k^*$. The property P_k is recognizable by a $(k+1)$-counter monitor where each counter keeps track of occurrences of one letter and the transitions are guarded with appropriate tests. We show that there is no k-counter monitor to recognize P_k, which gives us an expressiveness hierarchy based on the number of counters.

We consider several variants of counter monitors. In particular, we show that instruction sets $\langle +1, = \rangle$, $\langle +1, \geq \rangle$, and $\langle -1, +1, = 0 \rangle$ are equally expressive. A *copy update* features a register variable which occurs on the right-hand side of more than one assignment, and a *reset update* is an assignment of a register to 0. We also show that the *copyless* counter monitors are strictly less expressive than (copyful) counter monitors. Moreover, for every (copyful) counter monitor, there is a time-equivalent copyless reset-counter monitor.

Next, we focus on $\langle 1, +, = \rangle$-monitors, called *adder monitors*. The ability to compute sums of registers gives adder monitors dramatically more expressive power, and notably the ability to encode the prefixes of a word in real time, as shown in Figure 1. Also, we show that the language L in Figure 1 cannot be recognized by any counter monitor, establishing that adder monitors are strictly more expressive. However, interestingly, the language $L' = \bigcup_{n,m \in \mathbb{N}} \#(a^n \# \cup a^m \#)^\omega \cup \#(a^n \# \cup a^m \#)^* a^\omega$ can be recognized by a real-time 3-counter monitor but not by a real-time 2-adder monitor.

Considering the adder variants, we show that $\langle 1, +, = \rangle$-monitors and $\langle 1, +, -, = 0 \rangle$-monitors are equally expressive. Surprisingly, we show that in copyless monitors, adders are not more expressive than counters. In particular, any copyless reset-adder monitor with k registers can be simulated in real time by a (copyful) counter monitor with 2^k registers.

Beyond adders, we look at $\langle 0, 1, +, -, \geq \rangle$- and $\langle 0, 1, +, -, \times, \geq \rangle$-monitors, called *linear* and *polynomial monitors*, respectively. For every $k \in \mathbb{N}$, linear monitors with $4k$ registers can simulate Turing machines with k tapes in real-time. Despite their powerful instruction set, polynomial monitors cannot recognize the property $H = \#\Sigma'^\omega \setminus \bigcup_{w \in \Sigma^*} (\#\Sigma^*)^* \# w (\#\Sigma^*)^* \# w \# \Sigma'^\omega$, where $\Sigma = \{a, b\}$ and $\Sigma' = \Sigma \cup \{\#\}$. When finite words over Σ represent numbers in binary notation and $\#$ separates words into numbers, the property H represents sequences in which no number repeats.

4.2 Leveraging prior knowledge in monitoring [74]

We extend the classical definition of monitorability [88, 26] to account for assumptions. Intuitively, an assumption limits the universe of possible traces: when there are no assumptions the system can produce any trace in Σ^ω, but under an assumption A, all observed traces come from the set A. For a boolean property P, an assumption A, and a finite prefix s of some trace in A (denoted $s \in \mathit{Pref}(A)$), we say that P is *positively* (resp. *negatively*) *determined under* A by s iff, for every infinite extension f, if $sf \in A$ then $sf \in P$ (resp. $sf \notin P$). Then, we say that

- P is *s-monitorable under assumption* A iff there is a finite continuation r such that $sr \in \mathit{Pref}(A)$ positively or negatively determines P under A.
- P is *monitorable under* A iff it is s-monitorable under A for all finite traces $s \in \mathit{Pref}(A)$. We denote the set of properties that are monitorable under A by $\mathsf{Mon(A)}$.

Note that when $A = \Sigma^\omega$, our definitions are equivalent to the classical definition of monitorability.

We show that for every assumption A, the set $\mathsf{Mon(A)}$ is closed under boolean operations. However, when it comes to operations on assumptions themselves, the picture is drastically different. We demonstrate that monitorability is not preserved under complementation, intersection, nor under union of assumptions. Moreover, monitorability is neither downward nor upward preserved in general, but for each direction we identify a special case in which monitorability is preserved.

With a focus on monitorability, we extend the notion of relative safety [73] to finite boolean combinations of safety properties. We show that, for every assumption A, finite boolean combination of safety properties relative to A are monitorable under A.

Using the counter hierarchy described in Section 4.1, we show how assumptions can save resources. In particular, we prove that for every $k \geq 1$ and $1 \leq \ell \leq k$ there is a property P_k that is k-counter monitorable

under Σ^ω and a safety assumption A_ℓ such that A_ℓ is ℓ-counter monitorable under Σ^ω and P_k is $(k-\ell)$-counter monitorable under A_ℓ.

Finally, we describe ways to construct assumptions that make given properties safe, co-safe, or monitorable. The constructions are purely topological and rely on relativization. Let X be a topological space, and $S \subseteq X$ be a set. The set S is *closed* iff it contains all of its limit points. The complement of a closed set is *open*. The *closure* of S is the smallest closed set containing S, denoted $cl(S)$. Similarly, the *interior* of S is the largest open set contained in S, denoted $int(S)$. The *boundary* of S contains those points in the closure of S that do not belong to the interior of S, that is, $bd(P) = cl(P) \setminus int(P)$. The set S is *dense* iff every point in X is either in S or a limit point of S, that is, $cl(S) = X$. Similarly, S is *nowhere dense* iff $int(cl(S)) = \emptyset$. The safety properties correspond to the closed sets in the Cantor topology on Σ^ω, and the liveness properties correspond to the dense sets [8]. Moreover, the co-safety properties are the open sets [37], and the monitorable properties are the sets whose boundary is nowhere dense [49]. We show that for every property P there are unique weakest liveness assumptions A and B such that P is safe under A [73] and co-safe under B. Moreover, we show that for every property P there is a co-safety assumption A such that $P \in \mathsf{Mon}(\mathsf{A})$.

4.3 Monitoring quantitative properties precisely and approximately [75]

The main contribution of this work is the formalization of a framework for monitoring quantitative properties in which precision-resource trade-offs can be analyzed. A *quantitative property* is a function from Σ^ω to a value domain $\mathbb{D}$ which is a complete lattice. A *verdict* is a function from Σ^* to $\mathbb{D}$, and it represents how a monitor behaves. On an infinite trace, the verdict function yields an infinite verdict sequence. Since the monitor assigns a verdict value to every finite prefix of an infinite trace, we take the verdict value at the limit as the monitor's "estimate" for the property value. These estimates can under- or over-approximate property values which we capture by taking lim sup and lim inf of verdict sequences. Let us call the lim sup of a verdict sequence the *upper-estimate* provided by the verdict function, and the lim inf the *lower-estimate*. For a property p and a verdict v, we say that

- p is *universally monitorable from below* (resp. *from above*) by v iff for every infinite trace the upper-estimate (resp. lower-estimate) equals the property value.
- p *existentially monitorable from below* (resp. *from above*) by v iff (i) for every infinite trace the upper-estimate of v is at most (resp. at least) the property value, and (ii) for every finite trace there is an infinite extension on which the upper-estimate (resp. lower-estimate) equals the property value.
- p *approximately monitorable from below* (resp. *from above*) by v iff for every infinite trace the upper-estimate (resp. lower-estimate) is at most (resp. at least) the property value.

Considering two verdict functions v and u that monitor a property p from below (resp. from above), we say that v is *more precise* than u iff (i) for every infinite trace the upper-estimate (resp. lower-estimate) of v is greater (resp. less) than or equal to that of u, and (ii) there is an infinite trace for which the upper-estimate (resp. lower-estimate) of v is strictly greater (resp. less).

Monotonic verdict functions are of particular interest because the estimates they provide for a quantitative property value are always conservative and can improve in quality over time. We characterize the classes of properties that are universally monitorable by monotonic verdicts. Let p be a quantitative property and, for every $s \in \Sigma^*$, let $\nu_p(s) = \sup\{p(sf) \mid f \in \Sigma^\omega\}$ and $\mu_p(s) = \inf\{p(sf) \mid f \in \Sigma^\omega\}$. Similarly as for verdict functions, given an infinite trace, ν_p and μ_p produce infinite sequences. We say that p is *continuous* (resp. *co-continuous*) iff for every infinite trace the property value equals the limit value of the infinite sequence given by ν_p (resp. μ_p). Intuitively, the continuous and co-continuous properties constitute well-behaved sets of properties in the sense that, to monitor them, there is no need for speculation. For example, considering a continuous property, the least upper bound can only decrease after reading longer prefixes; therefore, a verdict function monitoring such a property can simultaneously be conservative and precise. We make this connection more explicit and show that continuous (resp. co-continuous) properties satisfy the desirable property of being universally monitorable by monotonically decreasing (resp. increasing) verdict functions.

Our framework conservatively generalizes several well-known definitions of boolean monitorability [88, 26, 53, 3] and the safety-progress hierarchy [37]. Table 1 below summarizes a part of these relations.

	Universally monitorable from below		Existentially monitorable from below	
$\mathbb{D}$	Mono. incr.	Unrestricted	Mono. incr.	Unrestricted
$\mathbb{B}$	$\emptyset$ or Σ^ω	Obl	$\emptyset$ or Σ^ω	Obl
$\mathbb{B}_\bot$	Safe $\cap$ CoSafe	Obl	Mon	at least Mon $\cup$ React
$\mathbb{B}_t$	CoSafe	Resp	any $P \subseteq \Sigma^\omega$	any $P \subseteq \Sigma^\omega$
$\mathbb{B}_f$	Safe	Pers	any $P \subseteq \Sigma^\omega$	any $P \subseteq \Sigma^\omega$

Table 1: Correspondence between classes of boolean properties and quantitative monitorability. The set of classically monitorable properties are denoted by Mon. Value domains are (i) $\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$ where $\mathtt{T}$ and $\mathtt{F}$ are incomparable, (ii) $\mathbb{B}_\bot = \mathbb{B} \cup \{\bot\}$ where $\bot < \mathtt{T}$ and $\bot < \mathtt{F}$, (iii) $\mathbb{B}_t = \{\mathtt{T}, \mathtt{F}\}$ where $\mathtt{F} < \mathtt{T}$, and (iv) $\mathbb{B}_f = \{\mathtt{T}, \mathtt{F}\}$ where $\mathtt{T} < \mathtt{F}$.

Finally, we extend our definition of register monitors in Section 4.1 with an output function that maps every configuration of the machine to a value in $\mathbb{D}$ and serves as a verdict function. In particular, we require that each output is a value stored in one of the registers, zero, or infinity. Then, we observe the relation between resources of a register monitor and its precision. We focus on the number of registers of a counter monitor and extend the counter hierarchy described in Section 4.1 to the quantitative setting. We show that for every $k > 1$ there is a quantitative property q_k such that (i) it requires k counters for universal monitoring, and (ii) for every monitor for p_k with $\ell < k$ counters, there is a more precise monitor with $\ell + 1$ counters. We generalize this theorem by extending P_k from k quantities to natural numbers and encoding them in binary. Specifically, we show that there exists a property q such that (i) no number of counters is sufficient for universal monitoring, and (ii) for every monitor for q with k counters, there is a more precise one with $k + 1$ counters.

Another resource subject to the trade-off is the register operations. For example, the ability to add two registers enables growing register values exponentially, while counting can grow only them linearly. It indicates that there are some properties that are universally monitorable by adder monitors and not by counter monitors. Although we do not explore this dimension in detail, we conjecture that many results from Section 4.1 are again relevant and can be extended to the quantitative setting.

5 Future Work

A theoretical framework that enables the analysis of precision-cost trade-offs in monitoring must address several aspects of monitors such as expressiveness, synthesis, decomposition, and refinement. Below we identify and discuss several relevant directions that stem from our research objectives and previous work. For each direction, we give examples of specific questions that we aim to answer.

Monitor resources. One of the central questions in RV is that of *monitorability*. The existing theories only broadly classify properties as monitorable or non-monitorable. We plan to take it a step further and systematically study how monitorability and monitor resources relate. Together with the real-time restriction, the use of integer-valued registers yields a rich resource theory. Our exploration is ongoing (see Section 4.1 or [58]) but far from complete. For example, we know that every additional counter register gives more expressive power in monitorability, in contrast to the theory of computability. Moreover, the security property "no number repeats" is not monitorable in real-time, even if the monitors can add and multiply. We aim to investigate relations of this nature more in-depth.

- Is there a real-time adder hierarchy similar to the counter hierarchy? In other words, is it true that for every $k \in \mathbb{N}$ there exists a real-time $(k + 1)$-adder monitor without an equivalent real-time k-adder monitor?
- What is the relation between monitor resources and how "quickly" a monitor reaches a positive or negative verdict? Is there a fine-grained hierarchy of properties regarding such time constraints?

- Is there a characterization of boolean properties that are monitorable by certain subclasses of register monitors? For example, which properties are monitorable by counter monitors and which by adders?
- How do we synthesize a "minimal" register monitor for a given boolean specification?

Quantitative monitoring. A quantitative property assigns a value to an infinite trace while a monitor does so to a finite prefix. One needs to bridge this gap to define what it means for a quantitative property to be monitored. We propose a generalization of existing notions of boolean monitorability to the quantitative setting (see Section 4.3 or [75]). Taking upper or lower limits of verdict sequences as "estimates" provided by monitors, we define several modalities of *quantitative monitorability* based on how property values and estimates relate. For example, the maximal response time of a server is *universally monitorable* because the estimates of a monitor that computes maximal response time over finite prefixes always converge to the property value. Note that the estimates of such a monitor are *monotonically increasing*. On the other hand, we need *non-monotonic* estimates for monitoring the average response time. Building on this framework, we aim to explore classes of monitorable quantitative properties and study the monitor synthesis problem.

- Which quantitative properties are universally monitorable by non-monotonic verdict functions?
- Which quantitative properties are universally monitorable by which register monitors? In other words, is there a characterization of functions that are universally monitorable by counters, adders, or more powerful monitors?
- Is there a topological characterization of quantitative properties that relates with our definitions of quantitative monitorability?
- Given a quantitative property p and a certain amount of resources (type of register operations and number of registers), how do we synthesize a register monitor that universally or existentially monitors p from below or above?

Approximate monitoring. Compared to boolean monitors, the quantitative counterparts can be *approximate* in more interesting ways as the value domain allows the *comparison* of estimates beyond true and false. Crucially, this enables the comparison of monitors in several dimensions. For example, a monitor can be (i) *more precise* than another by providing better estimates (see Section 4.3 or [75]), (ii) *faster* than another by approaching the property value more quickly, (iii) *more frugal* than another by using fewer resources, and (iv) *stronger* than another by making weaker assumptions about the system and the environment (see Section 4.2 or [74]). We aim to explore these dimensions and their relations to provide more insight into relevant trade-offs in monitor design.

- Can we strengthen our results on precision and number of counters? For example, is there a characterization of properties that yield a precision hierarchy as discussed in Section 4.3? Does such a hierarchy apply to register monitors with more powerful operations?
- How can we synthesize approximate monitors for a given specification? Instead of relative qualities, can we talk about "absolute" precision, speed, frugality, or strength? Moreover, how can we use these comparison dimensions for the synthesis of approximate monitors?
- What is the relation between monitor resources and how "quickly" a monitor reaches a "precise" verdict?
- How does precision and speed of a monitor relate? It seems that "faster" implies "at least as precise" but under which conditions do we have "faster" implies "more precise"?
- What does it mean that a quantitative property is monitorable under an assumption? What is the structure of assumptions that make properties monitorable more precisely with the same amount of resources? What kind of assumptions help make monitors more frugal? How do we synthesize a most precise monitor under an assumption? What is the weakest assumption that makes a property universally monitorable?

Enforcement monitors. So far, we only considered passive monitors that observe a system and report their verdicts. Instead, we can consider monitors that also take corrective action to steer the system away from undesired situations, which are called *enforcement* monitors. Such monitors are augmented with a correction function that enables modifying inputs and outputs of the system. A significant criterion in enforcement is *parsimony*: an enforcement monitor must modify as few data values as possible. Therefore, besides the dimensions that carry from passive monitors, an enforcement monitor can be *more parsimonious* than another by modifying fewer data values. We aim to extend our framework to capture the enforceability of quantitative properties and similar trade-offs as for passive monitors.

- What does it mean that a quantitative property is enforceable? How does the enforceability of a quantitative property relate with monitor resources? Which properties are enforceable at which degrees of precision or parsimony?
- Given a property, how can we design an enforcement monitor that is as parsimonious as possible, as precise as possible, as fast as possible, as frugal as possible, and as strong as possible?

Decentralized monitoring. Complex software systems often consist of multiple distributed components. System specifications then depend on the behaviors of these individual parts. For such systems, communication within the network is generally a bottleneck, which hinders the practicality of central monitors. Thus, we need a *decomposition* theory for approximate quantitative monitoring and enforcement. Beyond decomposition, we can study *assume-guarantee monitoring*, where a local enforcement monitor can assume that the others succeed in enforcing their policies and use it as an assumption to improve its precision, speed, frugality, or parsimony. We aim to establish a decomposition theory of monitors that addresses precision-cost trade-offs as well as a theory of assume-guarantee monitoring.

- Given a global specification S that is monitorable with a certain cost, is there a set of local specifications such that (i) they are equivalent to S in conjunction, and (ii) each local specification can be monitored with a smaller cost? Can we use operations beyond simple boolean combinations to capture asynchrony of components? Can we augment our machine model for this problem to capture the communication complexity in such systems?
- Does every enforceable distributed policy yield an assume-guarantee structure? If not, which policies do? Can we make non-enforceable policies enforceable under an assumption? If so, can this assumption be enforced itself?

References

[1] H. Abbas, Y. V. Pant, and R. Mangharam. Temporal logic robustness for general signal classes. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 45–56, 2019.

[2] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

[3] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. An operational guide to monitorability. In P. C. Ölveczky and G. Salaün, editors, *Software Engineering and Formal Methods*, pages 433–453, Cham, 2019. Springer International Publishing.

[4] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. The best a monitor can do. In C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[5] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. On Runtime Enforcement via Suppressions. In S. Schewe and L. Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[6] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir. On bidirectional runtime enforcement. In K. Peters and T. A. C. Willemse, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, pages 3–21, Cham, 2021. Springer International Publishing.

[7] S. Almagor, U. Boker, and O. Kupferman. Discounting in ltl. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 424–439. Springer, 2014.

[8] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181 – 185, 1985.

[9] R. Alur, L. DAntoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 13–22. IEEE, 2013.

[10] R. Alur, D. Fisman, K. Mamouras, M. Raghothaman, and C. Stanford. Streamable regular transductions. *Theoretical Computer Science*, 807:15–41, 2020.

[11] R. Alur, D. Fisman, and M. Raghothaman. Regular programming for quantitative properties of data streams. In *European Symposium on Programming*, pages 15–40. Springer, 2016.

[12] R. Alur, K. Mamouras, and C. Stanford. Automata-based stream processing. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[13] R. Alur, K. Mamouras, and C. Stanford. Modular quantitative monitoring. *Proc. ACM Program. Lang.*, 3(POPL), Jan. 2019.

[14] G. Audrito, R. Casadei, F. Damiani, V. Stolz, and M. Viroli. Adaptive distributed monitors of spatial properties for cyber–physical systems. *Journal of Systems and Software*, 175:110908, 2021.

[15] C. Baier, C. Dubslaff, and S. Klüppelholz. Trade-off analysis meets probabilistic model checking. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2014.

[16] A. Bakhirkin, T. Ferrère, T. Henzinger, and D. Nickovic. The first-order logic of signals. In *International Conference on Embedded Software (EMSOFT)*, 2018.

[17] H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In D. Giannakopoulou and D. Méry, editors, *FM 2012: Formal Methods*, pages 68–84, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] E. Bartocci. Sampling-based decentralized monitoring for networked embedded systems. *arXiv preprint arXiv:1308.5337*, 2013.

[19] D. Basin, M. Gras, S. Krstić, and J. Schneider. Scalable online monitoring of distributed systems. In J. Deshmukh and D. Ničković, editors, *Runtime Verification*, pages 197–220, Cham, 2020. Springer International Publishing.

[20] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu. Enforceable security policies revisited. *ACM Transactions on Information and System Security (TISSEC)*, 16(1):1–26, 2013.

[21] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu. Monitoring compliance policies over incomplete and disagreeing logs. In *International Conference on Runtime Verification*, pages 151–167. Springer, 2012.

[22] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu. Monitoring metric first-order temporal properties. *Journal of the ACM (JACM)*, 62(2):1–45, 2015.

[23] D. Basin, F. Klaedtke, and E. Zălinescu. Failure-aware runtime verification of distributed systems. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, volume 45, pages 590–603. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.

[24] A. Bauer and Y. Falcone. Decentralised ltl monitoring. *Formal Methods in System Design*, 48(1):46–93, 2016.

[25] A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *International Workshop on Runtime Verification*, pages 126–138. Springer, 2007.

[26] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4), Sept. 2011.

[27] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Proceedings of the Workshop on Foundations of Computer Security (FCS'02), Copenhagen, Denmark*. Citeseer, 2002.

[28] J. Baumeister, B. Finkbeiner, S. Schirmer, M. Schwenger, and C. Torens. Rtlola cleared for take-off: monitoring autonomous aircraft. In *International Conference on Computer Aided Verification*, pages 28–39. Springer, 2020.

[29] D. Beauquier, J. Cohen, and R. Lanotte. Security policies enforcement using finite and pushdown edit automata. *International journal of information security*, 12(4):319–336, 2013.

[30] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In A. Bouajjani and O. Maler, editors, *Computer Aided Verification*, pages 140–156, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[31] U. Boker, K. Chatterjee, T. A. Henzinger, and O. Kupferman. Temporal specifications with accumulative values. *ACM Transactions on Computational Logic (TOCL)*, 15(4):1–25, 2014.

[32] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, D. A. Rosenblueth, and C. Travers. Decentralized asynchronous crash-resilient runtime verification. In *27th International Conference on Concurrency Theory (CONCUR 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[33] B. Bonakdarpour, S. Navabpour, and S. Fischmeister. Time-triggered runtime verification. *Formal Methods in System Design*, 43(1):29–60, 2013.

[34] P. Bouyer, N. Markey, and R. M. Matteplackel. Averaging in ltl. In *International Conference on Concurrency Theory*, pages 266–280. Springer, 2014.

[35] T. Brázdil, K. Chatterjee, V. Forejt, and A. Kučera. Multigain: A controller synthesis tool for mdps with multiple mean-payoff objectives. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 181–187. Springer, 2015.

[36] P. Caspi and A. Benveniste. Toward an approximation theory for computerised control. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *Embedded Software*, pages 294–304, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[37] E. Chang, Z. Manna, and A. Pnueli. The safety-progress classification. In *Logic and Algebra of Specification*, pages 143–202. Springer, 1993.

[38] K. Chatterjee and L. Doyen. Energy and mean-payoff parity markov decision processes. In *International Symposium on Mathematical Foundations of Computer Science*, pages 206–218. Springer, 2011.

[39] K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 199–208. IEEE, 2009.

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

[41] K. Chatterjee, T. A. Henzinger, and J. Otop. Quantitative monitor automata. In *International Static Analysis Symposium*, pages 23–38. Springer, 2016.

[42] K. Chatterjee, T. A. Henzinger, and J. Otop. Nested weighted automata. *ACM Transactions on Computational Logic (TOCL)*, 18(4):1–44, 2017.

[43] F. Chen and G. Roşu. Parametric trace slicing and monitoring. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 246–261. Springer, 2009.

[44] C. Colombo and Y. Falcone. Organising ltl monitors over distributed systems with a global clock. *Formal Methods in System Design*, 49(1):109–158, 2016.

[45] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, and D. Thoma. Tessla: temporal stream-based specification language. In *Brazilian Symposium on Formal Methods*, pages 144–162. Springer, 2018.

[46] B. d'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. Lola: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 166–174. IEEE, 2005.

[47] L. M. Danielsson and C. Sánchez. Decentralized stream runtime verification. In B. Finkbeiner and L. Mariani, editors, *Runtime Verification*, pages 185–201, Cham, 2019. Springer International Publishing.

[48] S. Demri and R. Lazić. Ltl with the freeze quantifier and register automata. *ACM Transactions on Computational Logic (TOCL)*, 10(3):1–30, 2009.

[49] V. Diekert and M. Leucker. Topology, monitorable properties and runtime verification. *Theoretical Computer Science*, 537:29–41, 2014.

[50] A. Donzé, T. Ferrere, and O. Maler. Efficient robust monitoring for stl. In *International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.

[51] X. Du, Y. Liu, and A. Tiu. Trace-length independent runtime monitoring of quantitative policies in ltl. In *International Symposium on Formal Methods*, pages 231–247. Springer, 2015.

[52] Y. Falcone, T. Cornebize, and J.-C. Fernandez. Efficient and generalized decentralized monitoring of regular languages. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 66–83. Springer, 2014.

[53] Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012.

[54] Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem. Runtime verification of component-based systems in the bip framework with formally-proved sound and complete instrumentation. *Software & Systems Modeling*, 14(1):173–199, 2015.

[55] Y. Falcone, S. Krstic, G. Reger, and D. Traytel. A taxonomy for classifying runtime verification tools. In C. Colombo and M. Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science*, pages 241–262. Springer, 2018.

[56] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[57] P. Faymonville, B. Finkbeiner, S. Schirmer, and H. Torfah. A stream-based specification language for network monitoring. In *International Conference on Runtime Verification*, pages 152–168. Springer, 2016.

[58] T. Ferrère, T. A. Henzinger, and N. E. Saraç. A theory of register monitors. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 394–403, 2018.

[59] P. W. Fong. Access control by tracking shallow execution history. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 43–55. IEEE, 2004.

[60] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 112–127. Springer, 2011.

[61] P. Fraigniaud, S. Rajsbaum, and C. Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *International Conference on Runtime Verification*, pages 92–107. Springer, 2014.

[62] A. Francalanza. A theory of monitors. *Information and Computation*, page 104704, 2021.

[63] A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. Della Monica, and A. Ingólfsdóttir. A foundation for runtime monitoring. In S. Lahiri and G. Reger, editors, *Runtime Verification*, pages 8–29, Cham, 2017. Springer International Publishing.

[64] A. Francalanza, A. Gauci, and G. J. Pace. Distributed system contract monitoring. *The Journal of Logic and Algebraic Programming*, 82(5-7):186–215, 2013.

[65] A. Francalanza and A. Seychell. Synthesising correct concurrent runtime monitors. *Formal Methods in System Design*, 46(3):226–261, 2015.

[66] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *International Conference on Protocol Specification, Testing and Verification*, pages 3–18. Springer, 1995.

[67] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 412–416. IEEE, 2001.

[68] R. Grigore, D. Distefano, R. L. Petersen, and N. Tzevelekos. Runtime verification based on register automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 260–276. Springer, 2013.

[69] K. Havelund. Rule-based runtime verification revisited. *International Journal on Software Tools for Technology Transfer*, 17(2):143–170, 2015.

[70] K. Havelund and D. Peled. Runtime verification: From propositional to first-order temporal logic. In *International Conference on Runtime Verification*, pages 90–112. Springer, 2018.

[71] K. Havelund and G. Rosu. Monitoring programs using rewriting. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 135–143. IEEE, 2001.

[72] K. Havelund and G. Roşu. Synthesizing monitors for safety properties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer, 2002.

[73] T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43(3):135 – 141, 1992.

[74] T. A. Henzinger and N. E. Saraç. Monitorability under assumptions. In J. Deshmukh and D. Ničković, editors, *Runtime Verification*, pages 3–18, Cham, 2020. Springer International Publishing.

[75] T. A. Henzinger and N. E. Saraç. Quantitative and approximate monitoring. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2021.

[76] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković. Quantitative monitoring of stl with edit distance. *Formal methods in system design*, 53(1):83–112, 2018.

[77] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Computational analysis of run-time monitoring: Fundamentals of java-mac1 1this research was supported in part by onr n00014-97-1-0505, nsf ccr-9988409, nsf ccr-0086147, nsf cise-9703220, and aro daad19-01-1-0473. *Electronic Notes in Theoretical Computer Science*, 70(4):80 – 94, 2002. RV'02, Runtime Verification 2002 (FLoC Satellite Event).

[78] M. Kwiatkowska. Quantitative verification: Models techniques and tools. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, page 449–458, New York, NY, USA, 2007. Association for Computing Machinery.

[79] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and A. Schramm. Tessla: runtime verification of non-synchronized real-time streams. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1925–1933, 2018.

[80] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and D. Thoma. Runtime verification for timed event streams with partial information. In *International Conference on Runtime Verification*, pages 273–291. Springer, 2019.

[81] J. Ligatti, L. Bauer, and D. Walker. Enforcing non-safety security policies with program monitors. In *European Symposium on Research in Computer Security*, pages 355–373. Springer, 2005.

[82] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):1–41, 2009.

[83] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[84] E. Paul. Monitor logics for quantitative monitor automata. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[85] S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, A. Rollet, and O. N. Timo. Runtime enforcement of timed properties revisited. *Formal Methods in System Design*, 45(3):381–422, 2014.

[86] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa. Predictive runtime verification of timed properties. *Journal of Systems and Software*, 132:353–365, 2017.

[87] S. Pinisetty, V. Preoteasa, S. Tripakis, T. Jéron, Y. Falcone, and H. Marchand. Predictive runtime enforcement. *Formal Methods in System Design*, 51(1):154–199, 2017.

[88] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods*, pages 573–586, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[89] M. Rinard. Acceptability-oriented computing. *Acm sigplan notices*, 38(12):57–75, 2003.

[90] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[91] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings. 26th International Conference on Software Engineering*, pages 418–427. IEEE, 2004.

[92] C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement under memory-limitation constraints. *Information and Computation*, 206(2-4):158–184, 2008.