From LTL to rLTL Monitoring: Improved Monitorability through Robust Semantics

Corto Mascle
ENS Paris-Saclay
Paris, France
corto.mascle@ens-paris-saclay.fr

Daniel Neider
Max Planck Institute for Software Systems
Kaiserslautern, Germany
neider@mpi-sws.org

Maximilian Schwenger
Saarland University
Saarbrücken, Germany
schwenger@react.uni-saarland.de

Paulo Tabuada
UCLA
Los Angeles, USA
tabuada@ee.ucla.edu

Alexander Weinert
German Aerospace Center (DLR)
Cologne, Germany
alexander.weinert@dlr.de

Martin Zimmermann
University of Liverpool
Liverpool, United Kingdom
martin.zimmermann@liverpool.ac.uk

ABSTRACT

Runtime monitoring is commonly used to detect the violation of desired properties in safety critical cyber-physical systems by observing its executions. Bauer et al. introduced an influential framework for monitoring Linear Temporal Logic (LTL) properties based on a three-valued semantics: the formula is already satisfied by the given prefix, it is already violated, or it is still undetermined, i.e., it can still be satisfied and violated by appropriate extensions. However, a wide range of formulas are not monitorable under this approach, meaning that they have a prefix for which satisfaction and violation will always remain undetermined no matter how it is extended. In particular, Bauer et al. report that 44% of the formulas they consider in their experiments fall into this category.

Recently, a robust semantics for LTL was introduced to capture different degrees by wich a property can be violated. In this paper we introduce a robust semantics for finite strings and show its potential in monitoring: every formula considered by Bauer et al. is monitorable under our approach. Furthermore, we discuss which properties that come naturally in LTL monitoring — such as the realizability of all truth values — can be transferred to the robust setting. Lastly, we show that LTL formulas with robust semantics can be monitored by deterministic automata and report on a prototype implementation.

CCS CONCEPTS

• **Theory of computation** → **Formal languages and automata theory**; **Modal and temporal logics**; *Logic and verification*; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

KEYWORDS

Runtime Monitoring, Linear Temporal Logic, Robustness

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HSCC'20, April 22-24, 2020, Sydney, Australia
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7018-9/20/04... $15.00
https://doi.org/10.1145/3365365.3382197

ACM Reference Format:
Corto Mascle, Daniel Neider, Maximilian Schwenger, Paulo Tabuada, Alexander Weinert, and Martin Zimmermann. 2020. From LTL to rLTL Monitoring: Improved Monitorability through Robust Semantics. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'20).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3365365.3382197

1 INTRODUCTION

Runtime monitoring is nowadays routinely used to assess the satisfaction of properties of systems during their execution. To this end, a monitor, a finite-state device that runs in parallel to the system during deployment, evaluates it with respect to a fixed property. This is especially useful for systems that cannot be verified prior to deployment and, for this reason, can contain hidden bugs. While it is useful to catch and document these bugs during an execution of a system, we find that the current approach to runtime verification based on Linear Temporal Logic (LTL) [13] is not sufficiently informative, especially in what regards a system's robustness. Imagine that we are monitoring a property φ and that this property is violated during an execution. In addition to be alerted to the presence of a bug, there are several other questions we would like to have answered such as: Although φ was falsified, was there a *weaker* version of φ that was still satisfied or did the system fail catastrophically? Similarly, if we consider a property of the form $\varphi \rightarrow \psi$, where φ is an environment assumption and ψ is a system guarantee, and the environment violates φ *slightly* along an execution can we still guarantee that ψ is only *slightly* violated?

Answering these questions requires a logical formalism for specifying properties that provides meaning to terms such as *weaker* and *slightly*. Formalizing these notions within temporal logic, so as to be able to reason about the robustness of a system, was the main impetus behind the definition of *robust Linear-time Temporal Logic* (rLTL) [51]. While reasoning in LTL yields a binary result, rLTL adopts a five-valued semantics representing different *shades of violation*. Consider, for example, the specification $\Box a \rightarrow \Box b$ requiring that b is always satisfied provided a is always satisfied. In LTL, if the premise a is violated in a single position of the trace, then the specification is satisfied vacuously, eliminating all requirements on the system regarding $\Box b$. In this case, rLTL detects a mild violation of the premise and thus allows for a mild violation of the conclusion.

While recent work covers the synthesis [51] and verification problem [5, 6, 51] for rLTL, the runtime verification problem is yet to be addressed. Since runtime verification can only rely on finite traces by its nature, interesting theoretical questions open up for rLTL with finite semantics. On the practical side, the very same reasons that make runtime verification for LTL so useful also motivate the need for developing the framework proposed in this paper for rLTL runtime verification. To this end, we tackle the problem of evaluating a property over infinite traces based on a finite prefix similarly to Bauer et al. [13]. If the available information is insufficient to declare a specification violated or satisfied, the monitor reports a ?. This concept is applied to each degree of violation of the rLTL semantics. Thus, the rLTL monitor's verdict consists of four three-valued bits, as the rLTL semantics is based on four two-valued bits. Each bit represents a degree of violation of the specification in increasing order of severity.

As an example, consider an autonomous drone that may or may not be in a stable state[1]. The specification requires that it remains stable throughout the entire mission. However, if the take-off is shaky due to bad weather, the drone is unstable for the first couple of minutes. An LTL monitor thus jumps to the conclusion that the specification is violated whereas an rLTL monitor only reports a *partial* violation. As soon as the drone stabilizes, the LTL monitor does not indicate any improvement while the rLTL monitor refines its verdict to also report a partial satisfaction.

Some interesting properties that come naturally with LTL monitoring cannot be seamlessly lifted to rLTL monitoring. While it is obvious that all three truth values for finite trace LTL, i.e., satisfied, violated, and unknown, can be realized for some prefix and formula, the same does not hold for rLTL. Intuitively, the second and third bit of the rLTL monitor's four-bit output for the property $\boxdot a$ represent whether a eventually hold forever or whether it holds infinitely often, respectively. Based on a prefix, a monitor cannot distinguish between these two shades of violation, rendering some monitor outputs unrealizable.

In addition to that, we investigate how the level of informedness of an LTL monitor relates to the one of an rLTL monitor. The first observation is that a verdict of an LTL monitor can be refined at most once, from an unknown to either true or false. With rLTL semantics, however, a monitor can refine its output for a given formula up to four times. Secondly, an LTL monitor can only deliver meaningful verdicts for *monitorable* [12] properties. Intuitively, a property is monitorable if every prefix can be extended by a finite continuation that gives a definite verdict. We adapt the definition to robust monitoring and show that neither does LTL monitorability imply rLTL monitorability, nor vice versa.

Notwithstanding the above, empirical data suggests that rLTL monitoring indeed provides more information than LTL monitoring: This paper presents an algorithm synthesizing monitors for rLTL specifications. An implementation thereof allows us to validate the approach by replicating the experiments of Bauer et al. [12]. As performance metric, we use LTL and rLTL monitorability. They showed that 44% of their formulas are not LTL-monitorable whereas we show all of them to be rLTL-monitorable. This indicates that rLTL monitoring is an improvement over LTL monitoring in terms of monitorability and complements the theoretical results with a practical validation.

The main research contributions of this paper are thus a finite trace semantics for rLTL coupled with an investigation of its properties when compared to LTL, as well as an algorithm to synthesize monitors for rLTL specifications. The construction is doubly-exponential in the size of the formula, so rLTL monitoring is no more costly than LTL monitoring.

Proofs and detailed results of the experimental evaluation omitted due to space constraints can be found in the appendix of the technical report[2].

Related Work. In runtime verification [19, 31, 38, 44] the specification is often given in LTL [42]. While properties arguing about the past or current state of a system are always monitorable [30], LTL can also express assumptions on the future that cannot be validated using only a finite prefix of a word. Thus, adaptations of LTL have been proposed which include different notions of a next step on finite words [21, 41], lifting LTL to a three- or four-valued domain [12, 13], or applying predictive measures to rule out impossible extensions of words [53].

Non-binary monitoring has also been addressed by adding quantitative measures such as counting events [8, 43]. Most notably, Bartocci et al. [9] evaluate the "likelihood" that a satisfying or violating continuation will occur. To this end, for a given prefix, they count how long a continuation needs to be such that the specification is satisfied/violated; these numbers are then compared against each other. The resulting verdict is quinary: satisfying/violating, presumably satisfying/violating, or inconclusive. This approach is similar in nature to our work as it assesses the degree of satisfaction or violation of a given prefix. However, the motivation and niche of both approaches differs: Bartocci et al.'s approach computes — intuitively speaking — the amount of work that is required to satisfy or violate a specification, which allows for estimating the likelihood of satisfaction. Our approach, however, focusses on measuring to what an extend a specification was satisfied or violated.

Apart from that, monitoring tools collecting statistics [1, 4, 26] become increasingly popular: Snort [48] is a commercial tool for rule-based network monitoring and computing efficient statistics, Beep Beep 3 [29] is a tool based on a query language allowing for powerful aggregation functions and statistical measures. On the downside, it imposes the overhead of running a heavy-weight application on the monitored system. In contrast, we generate monitor automata out of an rLTL formula. Such an automaton can easily and automatically be implemented on almost any system with statically determined memory requirements and negligible performance overhead. Similarly, the Copilot [45] framework based on synchronous languages [15, 16] transforms a specification in a declarative data-flow language into a C implementation of a monitor with constant space and time requirements. Lola [2, 16] allows for more involved computations, also incorporating parametrization [25] and real-time capabilities [24] while retaining constant space and time requirements.

Another approach is to enrich temporal logics with quantitative measures such as taking either the edit distance [33], counting the

[1] By this we mean, e.g., that the error in tracking a desired trajectory is below a certain threshold.

[2] https://arxiv.org/abs/1807.08203

number of possible infinite models for LTL [27, 52], incorporating aggregation expressions into metric first-order temporal logic [10], or using averaging temporal operators that quantify the degree of satisfaction of a signal for a specification by integrating the signal w.r.t. a constant reference signal [3].

Rather than enriching temporal logics with such strong quantitative measures, we consider a robust version of LTL: rLTL [5, 6, 51]. Robust semantics yields information about to which degree a trace violates a property. We adapt the semantics to work with finite traces by allowing for intermediate verdicts. Here, a certain degree of violation can be classified as "indefinite" and refined when more information becomes available to the monitor. In a similar fashion, for Signal Temporal Logic [39, 40], Fainekos et al. [22] introduced a notion of spacial robustness based on interpreting atomic propositions over the real numbers. The sign of the real number provides information about satisfaction/violation while its absolute value provides information about robustness, i.e., how much can this value be altered without changing satisfaction/violation. This approach is complementary to ours since the notion of robustness in rLTL is related to the temporal evolution of atomic propositions which are interpreted classically, i.e, over the Booleans. Donze et al. [18] introduced a notion of robustness closer to rLTL in the sense that it measures how long we need to wait for the truth value of a formula to change. While the semantics of rLTL does not allow to quantify the exact delay needed to change the truth value of a formula, it allows to distinguish between the influence that different temporal evolutions, e.g., delays, persistence, and recurrence, have on the truth value of an LTL formula. Closer to rLTL is the work of Radionova et al. [47] (see also [50]) that established an unexpected connection between LTL and filtering through a quantitative semantics based on convolution with a kernel. By using different kernels one can express weaker or stronger interpretation of the same formula. However, this requires the user to choose multiple kernels and to use multiple semantics to reason about how the degradation of assumptions leads to the degradation of guarantees. In contrast, no such choices are required in rLTL.

2 ROBUST LINEAR TEMPORAL LOGIC

Throughout this work, we assume basic familiarity with classical LTL and refer the reader to a textbook for more details on the logic (see, e.g., [7]). Moreover, let us fix some finite set P of atomic propositions throughout the paper and define $\Sigma = 2^P$. We denote the set of finite and infinite words over Σ by Σ^* and Σ^ω, respectively. The empty word is denoted by ε and $\sqsubseteq$ and $\sqsubset$ denote the non-strict and the strict prefix relation, respectively. Moreover, we denote the set of Booleans by $\mathbb{B} = \{0, 1\}$.

The logics LTL and rLTL share the same syntax save for a dot superimposed on temporal operators. More precisely, the syntax of rLTL is given by the grammar

$$\begin{aligned}\varphi \coloneqq\ & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \rightarrow \varphi \\ & \mid \odot\varphi \mid \varphi\, \mathbf{U}\, \varphi \mid \varphi\, \mathbf{R}\, \varphi \mid \diamondsuit\varphi \mid \boxdot\varphi,\end{aligned}$$

where p ranges over atomic propositions in P and the temporal operators $\odot$, $\mathbf{U}$, $\mathbf{R}$, $\diamondsuit$ and $\boxdot$ correspond to "next", "until", "release", "eventually", and "always", respectively.[3] The size $|\varphi|$ of a formula φ is the number of its distinct subformulas. Furthermore, we denote the set of all LTL and rLTL formulas over P by Φ_{LTL} and Φ_{rLTL}, respectively.

The development of rLTL was motivated by the observation that the difference between "minor" and "major" violations of a formula cannot be adequately described in a two-valued semantics. If an LTL formula φ, for example, demands that the property p holds at all positions of a word $\sigma \in \Sigma^\omega$, then σ violates φ even if p does not hold at only a single position, a very minor violation. The semantics of LTL, however, does not differentiate between the σ above and a σ' in which the property p never holds, a major violation of the property φ.

In order to alleviate this shortcoming, Tabuada and Neider introduced Robust Linear-time Temporal Logic (rLTL) [51], whose semantics allows for distinguishing various "degrees" to which a word violates a formula. More precisely, the semantics of rLTL are defined over the set $\mathbb{B}_4 = \{0000, 0001, 0011, 0111, 1111\}$ of five *truth values*, each of which is a monotonically increasing sequence of four bits. We order the truth values in $\mathbb{B}_4$ by $0000 < 0001 < 0011 < 0111 < 1111$.

Intuitively, this order reflects increasingly desirable outcomes. If the specification is $\boxdot p$, the least desirable outcome, represented by 0000, is that p never holds on the entire trace. A slightly more desirable outcome is that p at least holds *sometime* but not infinitely often, which results in the value 0001. An even more desirable outcome would be if p holds infinitely often, while also being violated infinitely often, represented by 0011. Climbing up the ladder of desirable outcomes, the next best one requires p to hold infinitely often while being violated only finitely often, represented by the value 0111. Lastly, the optimal outcome fully satisfies $\Box p$, so p hold the entire time, represented by 1111. Thus, the first bit states whether $\Box p$ is satisfied, the second one stands for $\Diamond\Box p$, the third one for $\Box\Diamond p$, and the fourth one for $\Diamond p$. If all of them are 0, $\Box\neg p$ holds. The robust release is defined analogously.

The robust eventually-operator considers future positions in the trace and returns the truth value with the least degree of violation, which is a maximization with respect to the order defined above. This closely resembles the LTL definition. The robust until is defined analogously.

Based on this, the boolean conjunction and disjunction are defined as min and max, respectively, w.r.t. the order defined above, which generalizes the classical definition thereof. For the implication, consider a specification $\boxdot a \rightarrow \boxdot g$, where $\boxdot a$ is an assumption on the environment and $\boxdot g$ is a system guarantee. If the truth value of $\boxdot g$ is greater or equal to the one of $\boxdot a$, the assumption is fully satisfied. Thus, the rLTL semantics takes the violation of the assumption into account and lowers the requirements on the guarantees. However, if the guarantee exhibits a greater violation than the assumptions, the truth value of the implication is the same as the one of the guarantee. Lastly, the intuition behind the negation is that every truth value that is not 1111 constitutes a violation of the specification. Thus, the negation thereof is a full satisfaction (1111).

[3] Note that we include the operators $\wedge$, $\rightarrow$, and $\mathbf{R}$ explicitly in the syntax as they cannot be derived from other operators due to the many-valued nature of rLTL. Following the original work on rLTL [51], we also include the operators $\diamondsuit$ and $\boxdot$ explicitly (which can be derived from $\mathbf{U}$ and $\mathbf{R}$, respectively).

The negation of the truth value representing a perfect satisfaction (1111) is a full violation (0000).

To introduce the semantics, we need some additional notation: For $\sigma = \sigma(0)\sigma(1)\sigma(2)\cdots \in \Sigma^\omega$ and a natural number n, define $\sigma[n,\infty) = \sigma(n)\sigma(n+1)\sigma(n+2)\cdots$, (i.e., as the suffix of σ obtained by removing the first n letters of σ). To be able to refer to individual bits of an rLTL truth value $\beta \in \mathbb{B}_4$, we use $\beta[i]$ with $i \in \{1,\dots,4\}$ as to denote the i-th bit of β.

For the sake of a simpler presentation, we denote the semantics of both LTL and rLTL not in terms of satisfaction relations but by means of *valuation functions*. For LTL, the valuation function $V\colon \Sigma^\omega \times \Phi_{LTL} \to \mathbb{B}$ assigns to each infinite word $\sigma \in \Sigma^\omega$ and each LTL formula $\varphi \in \Phi_{LTL}$ the value 1 if σ satisfies φ and the value 0 if σ does not satisfy φ, and is defined as usual (see, e.g., [7]). The semantics of rLTL, on the other hand, is more complex and formalized next by an evaluation function $V_r\colon \Sigma^\omega \times \Phi_{rLTL} \to \mathbb{B}_4$ mapping an infinite word $\sigma \in \Sigma^\omega$ and an rLTL formula φ to a truth value in $\mathbb{B}_4$.

- $V_r(\sigma, p) = \begin{cases} 1111 & \text{if } p \in \sigma(0), \\ 0000 & \text{if } p \notin \sigma(0), \end{cases}$
- $V_r(\sigma, \neg\varphi) = \begin{cases} 1111 & \text{if } V_r(\sigma,\varphi) \neq 1111, \\ 0000 & \text{if } V_r(\sigma,\varphi) = 1111, \end{cases}$
- $V_r(\sigma, \varphi_1 \wedge \varphi_2) = \min\{V_r(\sigma,\varphi_1), V_r(\sigma,\varphi_2)\}$,
- $V_r(\sigma, \varphi_1 \vee \varphi_2) = \max\{V_r(\sigma,\varphi_1), V_r(\sigma,\varphi_2)\}$,
- $V_r(\sigma, \varphi_1 \rightarrow \varphi_2) = \begin{cases} 1111 & \text{if } V_r(\sigma,\varphi_1) \leq V_r(\sigma,\varphi_2), \\ V_r(\sigma,\varphi_2) & \text{if } V_r(\sigma,\varphi_1) > V_r(\sigma,\varphi_2), \end{cases}$
- $V_r(\sigma, \odot\varphi) = V_r(\sigma[1,\infty), \varphi)$,
- $V_r(\sigma, \diamondsuit\varphi) = \beta$ with
  $\beta[i] = \max_{n\geq 0} V_r(\sigma[n,\infty),\varphi)[i]$, $i \in \{1,\dots,4\}$,
- $V_r(\sigma, \boxdot\varphi) = \beta$ with
  - $\beta[1] = \min_{n\geq 0} V_r(\sigma[n,\infty),\varphi)[1]$,
  - $\beta[2] = \max_{m\geq 0} \min_{n\geq m} V_r(\sigma[n,\infty),\varphi)[2]$,
  - $\beta[3] = \min_{m\geq 0} \max_{n\geq m} V_r(\sigma[n,\infty),\varphi)[3]$,
  - $\beta[4] = \max_{n\geq 0} V_r(\sigma[n,\infty),\varphi)[4]$,
- $V_r(\sigma, \varphi_1 \,\mathbf{U}\, \varphi_2) = \beta$ with
  $\beta[i] = \max_{n\geq 0} \min\{\, V_r(\sigma[n,\infty),\varphi_2)[i],$
  $\min_{0\leq n'<n} V_r(\sigma[n',\infty),\varphi_1)[i] \,\}$, $i \in \{1,\dots,4\}$
- $V_r(\sigma, \varphi_1 \,\mathbf{R}\, \varphi_2) = \beta$ with
  - $\beta[1] = \min_{n\geq 0} \max\{V_r(\sigma[n,\infty),\varphi_2)[1],$
    $\max_{0\leq n'<n} V_r(\sigma[n',\infty),\varphi_1)[1]\}$,
  - $\beta[2] = \max_{m\geq 0} \min_{n\geq m} \max\{V_r(\sigma[n,\infty),\varphi_2)[2],$
    $\max_{0\leq n'<n} V_r(\sigma[n',\infty),\varphi_1)[2]\}$,
  - $\beta[3] = \min_{m\geq 0} \max_{n\geq m} \max\{V_r(\sigma[n,\infty),\varphi_2)[3],$
    $\max_{0\leq n'<n} V_r(\sigma[n',\infty),\varphi_1)[3]\}$, and
  - $\beta[4] = \max_{n\geq 0} \max\{V_r(\sigma[n,\infty),\varphi_2)[4],$
    $\max_{0\leq n'<n} V_r(\sigma[n',\infty),\varphi_1)[4]\}$.

Due to space restrictions, we have to refer the reader to the original work by Tabuada and Neider [51] for a thorough introduction and motivation.

However, we here want to illustrate the definition above and briefly argue that it indeed captures the intuition described at the beginning of this section. To this end, we reconsider the formulas $\boxdot p$ and $\boxdot a \rightarrow \boxdot g$ in Examples 2.1 and 2.2, respectively.

Example 2.1. Consider the formula $\boxdot p$ and the following five infinite words over the set $P = \{p\}$ of atomic propositions:

$\sigma_1 = \{p\}^\omega$	("p holds always")
$\sigma_2 = \emptyset\{p\}^\omega$	("p holds almost always")
$\sigma_3 = (\emptyset\{p\})^\omega$	("p holds infinitely often")
$\sigma_4 = \{p\}\emptyset^\omega$	("p holds finitely often")
$\sigma_5 = \emptyset^\omega$	("p holds never")

Let us begin the example with the word $\sigma_1 = \{p\}^\omega$. It is not hard to verify that $V_r(\sigma_1, \boxdot p)[1] = 1$ because p always holds in σ_1, i.e., $\min_{n\geq 0} V_r(\sigma[n,\infty),p)[1] = 1$ for $n \geq 0$. Using the same argument, we also have $V_r(\sigma_1, \boxdot p)[2] = V_r(\sigma_1, \boxdot p)[3] = V_r(\sigma_1, \boxdot p)[4] = 1$. Thus, $V_r(\sigma_1, \boxdot p) = 1111$.

Next, let us consider the word $\sigma_2 = \emptyset\{p\}^\omega$. In this case, we have $V_r(\sigma_1, \boxdot p)[1] = 0$ because $V_r(\sigma[0,\infty),p)[1] = 0$ (p does not hold in the first symbol of σ_2). However, $V_r(\sigma_1, \boxdot p)[2] = 1$ because p holds almost always, i.e., $\max_{m\geq 0} \min_{n\geq m} V_r(\sigma[n,\infty),p)[2] = 1$. Moreover, $V_r(\sigma_1, \boxdot p)[3] = V_r(\sigma_1, \boxdot a)[4] = 1$ and, thus, $V_r(\sigma_2, \boxdot p) = 0111$. Similarly, we obtain $V_r(\sigma_3, \boxdot p) = 0011$, $V_r(\sigma_4, \boxdot p) = 0001$, and $V_r(\sigma_5, \boxdot p) = 0000$.

In conclusion, this indeed illustrates that the semantics of the robust always is in accordance with the intuition provided at the beginning of this section. ⌟

Example 2.2. Let us now consider the more complex formula $\boxdot a \rightarrow \boxdot g$, where we interpret a to be an assumption on the environment of a cyber-physical system and g one of its guarantees. Moreover, let σ be an infinite word over $P = \{a, g\}$ such that $V_r(\sigma, \boxdot a \rightarrow \boxdot g) = 1111$. We now distinguish various cases.

First, let us assume that σ is such that $V_r(\sigma, \boxdot a) = 1111$, i.e., a always holds. By definition of the robust implication and since $V_r(\sigma, \boxdot a \rightarrow \boxdot g) = 1111$, this can only be the case if $V_r(\sigma, \boxdot g) = 1111$. Thus, the formula $\boxdot a \rightarrow \boxdot g$ ensures that if the environment assumption a always holds, so does the system guarantee g.

Next, assume that σ is such that $V_r(\sigma, \boxdot a) = 0111$, i.e., a does not always hold but almost always. By definition of the robust implication and since $V_r(\sigma, \boxdot a \rightarrow \boxdot g) = 1111$, this can only be the case if $V_r(\sigma, \boxdot g) \geq 0111$. In this case, the formula $\boxdot a \rightarrow \boxdot g$ ensures that if the environment assumption a holds almost always, then the system guarantee g holds almost always or—even better—always.

It is not hard to verify that we obtain similar results for the cases $V_r(\sigma, \boxdot a) \in \{0011, 0001, 0000\}$. In other words, the semantics of rLTL ensures that the violation of the system guarantee g is always proportional to the violation of the environment assumption a (given that $V_r(\sigma, \boxdot a \rightarrow \boxdot g)$ evaluates to 1111). Again, this illustrates that the semantics of the implication is in accordance with the intuition provided at the beginning of this section. ⌟

Alternatively, the rLTL evaluation function can be expressed by evaluating a sequence of four LTL formulas φ_i, one for each bit i, obtained by a straightforward rewriting of φ. Such a transformation for general formulas is explained in the appendix of the technical report.). The remark below illustrates how this is done for "simple" rLTL formulas.

Remark. *Let φ be an rLTL formula that has no always in the scope of a negation and only uses negation, conjunction, disjunction, next, eventually, and always. Then, for all $\sigma \in \Sigma^\omega$,*

- $V_r(\sigma, \varphi)[1] = V(\sigma, \varphi_1)$ *where φ_1 is the formula obtained from φ by replacing every $\odot/\diamondsuit/\boxdot$ by $\bigcirc/\diamond/\Box$, respectively,*
- $V_r(\sigma, \varphi)[2] = V(\sigma, \varphi_2)$ *where φ_2 is the formula obtained from φ by replacing every $\odot/\diamondsuit$ by $\bigcirc/\diamond$, and $\boxdot$ by $\diamond\Box$,*
- $V_r(\sigma, \varphi)[3] = V(\sigma, \varphi_3)$ *where φ_3 is the formula obtained from φ by replacing every $\odot/\diamondsuit$ by $\bigcirc/\diamond$, and $\boxdot$ by $\Box\diamond$,*
- $V_r(\sigma, \varphi)[4] = V(\sigma, \varphi_4)$ *where φ_4 is the formula obtained from φ by replacing every $\odot/\diamondsuit$ by $\bigcirc/\diamond$, and $\boxdot$ by $\diamond$.*

Finally, note that rLTL is an extension of LTL. In fact, the LTL semantics can be recovered from the first bit of the rLTL semantics [51] (after every implication $\varphi \rightarrow \psi$ has been replaced with $\neg\varphi \vee \psi$).[4]

3 MONITORING ROBUST LTL

In their work on LTL monitoring, Bauer et al. [13] define the problem of runtime monitoring as *"check[ing] LTL properties given finite prefixes of infinite [words]"*. More formally, given some prefix $u \in \Sigma^*$ and some LTL formula φ, it asks whether all, some, or no infinite extension $u\sigma \in \Sigma^\omega$ of u by some $\sigma \in \Sigma^\omega$ satisfies φ. To reflect these three possible results, the authors use the set $\mathbb{B}^? = \{0, ?, 1\}$ to define a three-valued logic that is syntactically identical to LTL, but equipped with a semantics in form of an evaluation function $V^m \colon \Sigma^* \times \Phi_{LTL} \rightarrow \mathbb{B}^?$ over finite prefixes. This semantics is defined such that $V^m(u, \varphi)$ is equal to 0 (is equal to 1) if no (if every) extension $u\sigma$ of u satisfies φ. If neither is the case, i.e., if there is an extension of u that satisfies φ and there is an extension of u that does not satisfy φ, then $V^m(u, \varphi)$ is equal to ?.

We aim to extend the approach of Bauer et al. to rLTL, whose semantics is based on truth values from the set $\mathbb{B}_4$ (containing the sequences of length four in 0^*1^*). As a motivating example, let us consider the formula $\varphi = \boxdot s$ for some atomic proposition s and study which situations can arise when monitoring this formula. Note that the truth value of φ can be obtained by concatenating the truth values of the LTL formulas $\varphi_1 = \Box s$, $\varphi_2 = \diamond\Box s$, $\varphi_3 = \Box\diamond s$, and $\varphi_4 = \diamond s$.

First, consider the empty prefix and its two extensions $\emptyset^\omega$ and $\{s\}^\omega$. We have $V_r(\emptyset^\omega, \varphi) = 0000$ and $V_r(\{s\}^\omega, \varphi) = 1111$. Thus, all four bits can both be equal to 0 and 1. This situation is captured by the sequence ???? which signifies that for every position i and every bit $b \in \mathbb{B}$, there exists an extension of ε that has bit b in the i-th position of the truth value with respect to φ.

Now, consider the prefix $\{s\}$ for which we have $V_r(\{s\}\sigma, \varphi)[4] = 1$ for every $\sigma \in \Sigma^\omega$ as $\varphi_4 = \diamond s$ is satisfied on each extension of $\{s\}$ (s has already occurred). On the other hand, $V_r(\{s\}\emptyset^\omega, \varphi) = 0001$ and $V_r(\{s\}\{s\}^\omega, \varphi) = 1111$, i.e., the first three bits can both be 0 and 1 by picking an appropriate extension. Hence, the situation is captured by the sequence ???1, signifying that the last bit is determined by the prefix, but the first three are not. Using dual arguments, the sequence 0??? is used for the prefix $\emptyset$, signifying that the first bit is determined by the prefix as every extension violates $\varphi_1 = \Box s$. However, the last three bits are not yet determined by the prefix, hence the trailing ?'s.

Finally, consider the prefix $\{s\}\emptyset$. Using the same arguments as for the previous two prefixes, we obtain $V_r(\{s\}\emptyset\sigma, \varphi)[1] = 0$ and $V_r(\{s\}\emptyset\sigma, \varphi)[4] = 1$ for every $\sigma \in \Sigma^\omega$. Also, as before, we have $V_r(\{s\}\emptyset\emptyset^\omega, \varphi) = 0001$ and $V_r(\{s\}\emptyset\{s\}^\omega, \varphi) = 0111$. Hence, here we obtain the sequence 0??1 signifying that the first and last bit are determined by the prefix, but the middle two are not.

In general, we use truth values of the form $0^*?^*1^*$, which follows from the fact that the truth values of rLTL are in 0^*1^*. Hence, let $\mathbb{B}_4^?$ denote the set of sequences of length four in $0^*?^*1^*$. Based on $\mathbb{B}_4^?$, we now formally define the rLTL monitoring semantics as a bitwise generalization of the LTL definition.

Definition 3.1. The semantics of the robust monitor $V_r^m \colon \Sigma^* \times \Phi_{rLTL} \rightarrow \mathbb{B}_4^?$ is defined as $V_r^m(u, \varphi) = \beta$ with

$$\beta[i] = \begin{cases} 0 & \text{if } V_r(u\sigma, \varphi)[i] = 0 \text{ for all } \sigma \in \Sigma^\omega; \\ 1 & \text{if } V_r(u\sigma, \varphi)[i] = 1 \text{ for all } \sigma \in \Sigma^\omega; \text{ and} \\ ? & \text{otherwise,} \end{cases}$$

for every $i \in \{1, \ldots, 4\}$, every rLTL formula φ, and every $u \in \Sigma^*$.

Using this semantics, we are able to infer information about the infinite run of a system after having read only a finite prefix thereof. In fact, this robust semantics provides far more information about the degree of violation of the specification than classical LTL monitoring as each bit of the monitoring output represents a degree of violation of the specification: a ? turning into a 0 or 1 indicates a deterioration or improvement in the system's state, respectively. Consider, for instance, an autonomous drone with specification $\varphi = \boxdot s$ where s denotes a state of stable flight (recall the motivating example on Page 5). Initially, the monitor would output ???? due to a lack of information. If taking off under windy conditions, the state s is not reached initially, hence the monitor issues a warning by producing $V_r^m(\emptyset^n, \varphi) = 0???$ for every $n > 0$. Thus, the safety condition is violated temporarily, but not irrecoverably. Hence, mitigation measures can be initiated. Upon success, the monitoring output turns into $V_r^m(\emptyset^n\{s\}, \varphi) = 0??1$ for every $n > 0$, signaling that flight was stable for some time.

Before we continue, let us first state that the new semantics is well-defined, i.e., that the sequence $\beta[1]\beta[2]\beta[3]\beta[4]$ in Definition 3.1 is indeed in $\mathbb{B}_4^?$.

Lemma 3.2. *$V_r^m(u, \varphi) \in \mathbb{B}_4^?$ for every rLTL formula φ and every $u \in \Sigma^*$.*

After having shown that every possible output of V_r^m is in $\mathbb{B}_4^?$, the next obvious question is whether V_r^m is surjective, i.e., whether every truth value $\beta \in \mathbb{B}_4^?$ is *realized* by some prefix $u \in \Sigma^*$ and some rLTL formula φ in the sense that $V_r^m(u, \varphi) = \beta$. Recall the motivating example above: The formula $\boxdot s$ realizes at least the following four truth values: ???? (on ε), ???1 (on $\{s\}$), 0??? (on $\emptyset$), and 0??1 (on $\{s\}\emptyset$). It is not hard to convince oneself that these are all truth values realized by $\boxdot s$ as they represent the following four types of prefixes that can be distinguished: the prefix is empty

[4] It turns out that Tabuada and Neider's original proof [51, Proposition 5] has a minor mistake. Although the first bit of the rLTL semantics coincides with the original LTL semantics for all formulas that do not contain implications, the formula $\boxdot\neg a \rightarrow \boxdot a$ is an example witnessing this claim is no longer correct in the presence of implications, e.g., for $\{a\}\emptyset^\omega$. However, this issue can be fixed by replacing every implication $\varphi \rightarrow \psi$ with $\neg\varphi \vee \psi$. This substitution results in an equivalent LTL formula for which the first bit of the rLTL semantics indeed coincides with the LTL semantics.

(truth value ????), the prefix is in $\{s\}^+$ (truth value ???1), the prefix is in $\emptyset^+$ (truth value 0???), or the prefix contains both an $\{s\}$ and an $\emptyset$ (truth value 0??1).

For most other truth values, it is straightforward to come up with rLTL formulas and prefixes that realize them. See Table 1 for an overview and recall Remark 2, which is applicable to all these formulas.

For others, such as 0011, it is much harder. Intuitively, to realize 0011, one needs to find an rLTL formula φ and a prefix $u \in \Sigma^*$ such that the formula obtained by replacing all $\boxdot$ in φ by $\Diamond\Box$ is not satisfied by any extension of u, but the formula obtained by replacing all $\boxdot$ in φ by $\Box\Diamond$ is satisfied by every extension of u.[5] Thus, intuitively, the prefix has to differentiate between a property holding almost always and holding infinitely often. It turns out that no such u and φ exist. A similar argument is true for 0001, leading to the following theorem.

Theorem 3.3. *All truth values except for* 0011 *and* 0001 *are realizable.*

As shown in Table 1, all of the realizable truth values except for 0111 are realized by formulas using only conjunction, disjunction, negation, eventually, and always. Further, 0111 can only be realized by a formula with the release operator while the truth values 0011 and 0001 are indeed not realizable at all.

Note that the two unrealizable truth values 0011 and 0001 both contain a 0 that is directly followed by a 1. The proof of unrealizability formalizes the intuition that such an "abrupt" transition from definitive violation of a property to definitive satisfaction of the property cannot be witnessed by any finite prefix. Finally, the only other truth value of this form, 0111, is only realizable by using a formula with the release operator.

Going again back to the motivating example $\boxdot s$, consider the evolution of the truth values on the sequence $\varepsilon, \{s\}, \{s\}\emptyset$: They are ????, ???1, and 0??1, i.e., 0's and 1's are stable when extending a prefix, only a ? may be replaced by a 0 or a 1. This property holds in general. To formalize this, say that $\beta' \in \mathbb{B}_4^?$ is more specific than $\beta \in \mathbb{B}_4^?$, written as $\beta \preceq \beta'$, if, for all i, $\beta[i] \neq ?$ implies $\beta'[i] = \beta[i]$.

Lemma 3.4. *Let φ be an rLTL formula and $u, u' \in \Sigma^*$. If $u \sqsubseteq u'$, then $V_r^m(u, \varphi) \preceq V_r^m(u', \varphi)$.*

Lemma 3.4 immediately implies two properties of the semantics: *impartiality* and *anticipation* [17]. Impartiality states that a definitive verdict will never be revoked: If $V_r^m(u, \varphi)[i] \neq ?$, then for all finite extensions $v \in \Sigma^*$, the verdict will not change, so $V_r^m(uv, \varphi)[i] = V_r^m(u, \varphi)[i]$. Anticipation requires that a definitive verdict is decided as soon as possible, i.e., if $V_r^m(u, \varphi)[i] = ?$, then u can still be extended to satisfy and to violate φ with the i-th bit. Formally, there have to exist infinite extensions σ_0 and σ_1 such that $V_r(u\sigma_0, \varphi)[i] = 0$ and $V_r(u\sigma_1, \varphi)[i] = 1$. Anticipation holds by definition of $V_r^m(u, \varphi)$.

Due to Lemma 3.4, for a fixed formula, the prefixes of every infinite word can assume at most five different truth values, which are all of increasing specificity. It turns out that this upper bound is tight. To formalize this claim, we denote the strict version of $\preceq$ by $\prec$, i.e., $\beta \prec \beta'$ if and only if $\beta \preceq \beta'$ and $\beta \neq \beta'$.

[5]Note that this intuition breaks down in the presence of implications and negation, due to their non-standard definitions.

Lemma 3.5. *There is an rLTL formula φ and prefixes $u_0 \sqsubset u_1 \sqsubset u_2 \sqsubset u_3 \sqsubset u_4$ such that $V_r^m(u_0, \varphi) \prec V_r^m(u_1, \varphi) \prec V_r^m(u_2, \varphi) \prec V_r^m(u_3, \varphi) \prec V_r^m(u_4, \varphi)$.*

After determining how many different truth values can be assumed by prefixes of a single infinite word, an obvious question is how many truth values can be realized by a fixed formula on *different* prefixes. It is not hard to combine the formulas in Table 1 to a formula that realizes all truth values not ruled out by Theorem 3.3.[6]

Lemma 3.6. *There is an rLTL formula φ such that for every $\beta \in \mathbb{B}_4^? \setminus \{0011, 0001\}$ there is a prefix u_β with $V_r^m(u_\beta, \varphi) = \beta$.*

Finally, let us consider the notion of *monitorability* [46], an important concept in the theory of runtime monitoring. As a motivation, consider the LTL formula $\psi = \Box\Diamond s$ and an arbitrary prefix $u \in \Sigma^*$. Then, the extension $u\{s\}^\omega$ satisfies ψ while the extension $u\emptyset^\omega$ does not satisfy ψ, i.e., satisfaction of ψ is independent of any prefix u. Hence, we have $V^m(u, \psi) = ?$ for every prefix u, i.e., monitoring the formula ψ does not generate any information.

In general, for a fixed LTL formula φ, a prefix $u \in \Sigma^*$ is called *ugly* if we have $V^m(uv, \varphi) = ?$ for every finite $v \in \Sigma^*$, i.e., every finite extension of u yields an indefinite verdict.[7] Now, φ is *LTL-monitorable* if there is no ugly prefix with respect to φ. A wide range of LTL formulas (e.g., $\psi = \Box\Diamond s$ as above) are unmonitorable in that sense. In particular, 44% of the LTL formulas considered in the experiments of Bauer et al. are not LTL-monitorable.

We next generalize the notion of monitorability to rLTL. In particular, we answer whether there are unmonitorable rLTL formulas. Then, in Section 5, we exhibit that all LTL formulas considered by Bauer et al.'s experimental evaluation, even the unmonitorable ones, are monitorable under rLTL semantics. To conclude the motivating example, note that the rLTL analogue $\boxdot\Diamond\!\!\!\cdot\, s$ of the LTL formula ψ induces two truth values from $\mathbb{B}_4^?$ indicating whether s has been true at least once (truth value ???1) or not (truth value ????). Even more so, every prefix inducing the truth value ???? can be extended to one inducing the truth value ???1.

Definition 3.7. Let φ be an rLTL formula. A prefix $u \in \Sigma^*$ is called *ugly* if we have $V^m(uv, \varphi) = ????$ for every finite $v \in \Sigma^*$. Further, φ is *rLTL-monitorable* if it has no ugly prefix.

As argued above, the formula $\boxdot\Diamond\!\!\!\cdot\, s$ has no ugly prefix, i.e., it is rLTL-monitorable. Thus, we have found an unmonitorable LTL formula whose rLTL analogue (the formula obtained by adding dots to all temporal operators) is monitorable. The converse statement is also true. There is a monitorable LTL formula whose rLTL analogue is unmonitorable. To this end, consider the LTL formula

$$(\Box s \wedge \Box\neg s) \rightarrow (\Diamond\Box s \wedge \Diamond\neg\Diamond s),$$

which is a tautology and therefore monitorable. On the other hand, we claim that $\emptyset\{s\}$ is an ugly prefix for the rLTL analogue φ obtained

[6]However, there are also formulas assuming *many* truth values, e.g., the following formula from LTLStore [34], which assumes ten different truth values:

$$(((a \wedge d) \vee (\neg a \wedge \neg d)) \wedge \boxdot(\neg b \vee (\neg a \wedge d)))\vee$$
$$(((\neg a \wedge d) \vee (a \wedge \neg d)) \wedge \Diamond\!\!\!\cdot(b \wedge (a \vee \neg d))) \vee (a \wedge \boxdot b)$$

[7]Note that the good/bad prefixes introduced by Kupfermann and Vardi [36] always yield a positive/negative verdict, respectively. Ugly prefixes [13] always yield an indefinite verdict.

Table 1: Realizable truth values. For every truth value β, the next two columns show prefixes u and formulas φ such that $V_r^m(u, \varphi) = \beta$, or that β is unrealizable.

Value	*Prefix*	*Formula*	*Value*	*Prefix*	*Formula*
0000	ε	$a \wedge \neg a$	0?11	$\emptyset\{a\}$	$\boxdot a \vee \boxdot \neg a$
000?	ε	$\diamondsuit\boxdot a \wedge \diamondsuit \neg \diamondsuit a$	0111	$\emptyset\{a\}$	$a \,\mathbf{R}\, a$
0001	unrealizable		????	ε	$\boxdot a$
00??	ε	$\boxdot a \wedge \boxdot \neg a$	???1	$\{a\}$	$\boxdot a$
00?1	$\emptyset\{a\}$	$\boxdot a \wedge \boxdot \neg a$	??11	ε	$\boxdot a \vee \diamondsuit \neg \diamondsuit a$
0011	unrealizable		?111	ε	$\boxdot a \vee \neg \diamondsuit \neg \diamondsuit \neg a$
0???	$\emptyset$	$\boxdot a$	1111	ε	$a \vee \neg a$
0??1	$\emptyset\{a\}$	$\boxdot a$			

by adding dots to the temporal operators. To this end note that we have both $V_r(\emptyset\{s\}v\emptyset^\omega, \varphi) = 1111$ and $V_r(\emptyset\{s\}v\{s\}^\omega, \varphi) = 0000$ for every $v \in \Sigma^*$. Hence, $V_r^m(\emptyset\{s\}v, \varphi) = ????$ for every such v, i.e., $\emptyset\{s\}$ is indeed ugly and φ therefore not rLTL-monitorable.

Thus, there are formulas that are unmonitorable under LTL semantics, but monitorable under rLTL semantics and there are formulas that are unmonitorable under rLTL semantics, but monitorable under LTL semantics. Using these formulas one can also construct a formula that is unmonitorable under both semantics.

To this end, fix LTL formulas φ_ℓ and φ_r over disjoint sets of propositions and a fresh proposition p not used in either formula such that

- φ_ℓ has an ugly prefix u_ℓ under LTL semantics, and
- φ_r (with dotted operators) has an ugly prefix u_r under rLTL semantics.

We can assume both prefixes to be non-empty, as ugliness is closed under finite extensions. Let $\varphi = (p \wedge \varphi_\ell) \vee (\neg p \wedge \varphi_r)$. Then, the prefix obtained from u_ℓ by adding the proposition p to the first letter is ugly for φ under LTL semantics and u_r is ugly for φ (with dotted operators) under rLTL semantics.

As a final example, recall that we have shown that $\boxdot\diamondsuit s$ is rLTL-monitorable and consider its negation $\neg\boxdot\diamondsuit s$. It is not hard to see that $V_r^m(u, \varphi) = ????$ holds for every prefix u. Hence, ε is an ugly prefix for the formula, i.e., we have found another unmonitorable rLTL formula. In particular, the example shows that, unlike for LTL, rLTL-monitorability is not preserved under negation.

After having studied properties of rLTL monitorability, we next show our main result: The robust monitoring semantics V_r^m can be implemented by finite-state machines.

4 CONSTRUCTION OF RLTL MONITORS

An rLTL monitor is an implementation of the robust monitoring semantics V_r^m in form of a finite-state machine with output. More precisely, an *rLTL monitor* for an rLTL formula φ is a finite-state machine $\mathcal{M}_\varphi$ that on reading an input $u \in \Sigma^*$ outputs $V_r^m(u, \varphi)$. In this section, we show how to construct rLTL monitors and that this construction is asymptotically not more expensive than the construction of LTL monitors. Let us fix an rLTL formula φ for the remainder of this section.

Our rLTL monitor construction is inspired by Bauer et al. [13] and generates a sequence of finite-state machines (i.e., Büchi automata over infinite words, (non)deterministic automata over finite words, and Moore machines). Underlying these machines are *transition structures* $\mathcal{T} = (Q, q_I, \Delta)$ consisting of a nonempty, finite set Q of states, an initial state $q_I \in Q$, and a transition relation $\Delta \subseteq Q \times \Sigma \times Q$. An (infinite) run of $\mathcal{T}$ on a word $\sigma = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ is a sequence $\rho = q_0 q_1 \cdots$ of states such that $q_0 = q_I$ and $(q_j, a_j, q_{j+1}) \in \Delta$ for $j \in \mathbb{N}$. Finite runs on finite words are defined analogously. The transition structure $\mathcal{T}$ is *deterministic* if (a) $(q, a, q') \in \Delta$ and $(q, a, q'') \in \Delta$ imply $q' = q''$ and (b) for each $q \in Q$ and $a \in \Sigma$ there exists a $(q, a, q') \in \Delta$. We then replace the transition relation Δ by a function $\delta \colon Q \times \Sigma \to Q$. Finally, we define the *size* of a transition structure $\mathcal{T}$ as $|\mathcal{T}| = |Q|$ in order to measure its complexity.

Our construction then proceeds in three steps:

(1) We bring φ into an operational form by constructing Büchi automata $\mathcal{A}_\beta^\varphi$ for each truth value $\beta \in \mathbb{B}_4$ that can decide the valuation $V_r(\sigma, \varphi)$ of infinite words $\sigma \in \Sigma^\omega$.
(2) Based on these Büchi automata, we then construct nondeterministic automata $\mathcal{B}_\beta^\varphi$ that can decide whether a finite word $u \in \Sigma^*$ can still be extended to an infinite word $u\sigma \in \Sigma^\omega$ with $V_r(u\sigma, \varphi) = \beta$.
(3) We determinize the nondeterministic automata obtained in Step 2 and combine them into a single Moore machine that computes $V_r^m(u, \varphi)$.

Let us now describe each of these steps in detail.

Step 1: We first translate the rLTL formula φ into several Büchi automata using a construction by Tabuada and Neider [51], summarized in Theorem 4.1 below. A Büchi automaton is a four-tuple $\mathcal{A} = (Q, q_I, \Delta, F)$ where $\mathcal{T} = (Q, q_I, \Delta)$ is a transition structure and $F \subseteq Q$ is a set of accepting states. A run π of $\mathcal{A}$ on $\sigma \in \Sigma^\omega$ is a run of $\mathcal{T}$ on σ, and we say that π is accepting if it contains infinitely many states from F. The automaton $\mathcal{A}$ accepts a word σ if there exists an accepting run of $\mathcal{A}$ on σ. The language $\mathcal{L}(\mathcal{A})$ is the set of all words accepted by $\mathcal{A}$, and the size of $\mathcal{A}$ is defined as $|\mathcal{A}| = |\mathcal{T}|$.

Theorem 4.1 (Tabuada and Neider [51]). *Given a truth value $\beta \in \mathbb{B}_4$, one can construct a Büchi automaton $\mathcal{A}_\beta^\varphi$ with $2^{O(|\varphi|)}$ states*

such that $\mathcal{L}(\mathcal{A}^{\varphi}_{\beta}) = \{\sigma \in \Sigma^{\omega} \mid V_r(\sigma, \varphi) = \beta\}$. *This construction can be performed in* $2^{O(|\varphi|)}$ *time.*

The Büchi automata $\mathcal{A}^{\varphi}_{\beta}$ for $\beta \in \mathbb{B}_4$ serve as building blocks for the next steps.

Step 2: For each Büchi automaton $\mathcal{A}^{\varphi}_{\beta}$ obtained in the previous step, we now construct a nondeterministic automaton $\mathcal{B}^{\varphi}_{\beta}$ over finite words. This automaton determines whether a finite word $u \in \Sigma^*$ can be continued to an infinite word $u\sigma \in \mathcal{L}(\mathcal{A}^{\varphi}_{\beta})$ (i.e., $V_r(u\sigma, \varphi) = \beta$) and is used later to construct the rLTL monitor.

Formally, a *nondeterministic finite automaton (NFA)* is a four-tuple $\mathcal{A} = (Q, q_I, \Delta, F)$ that is syntactically identical to a Büchi automaton. The size of $\mathcal{A}$ is defined analogously to Büchi automata. In contrast to Büchi automata, however, NFAs only admit finite runs on finite words, i.e., a run of $\mathcal{A}$ on $u = a_0 \cdots a_{n-1} \in \Sigma^*$ is a sequence $q_0 \cdots q_n$ such that $q_0 = q_I$ and $(q_j, a_j, q_{j+1}) \in \Delta$ for every $j < n$. A run $q_0 \cdots q_n$ is called *accepting* if $q_n \in F$. Accepted words as well as the language of $\mathcal{A}$ are again defined analogously to the Büchi case. If (Q, q_I, Δ) is deterministic, $\mathcal{A}$ is a *deterministic finite automaton (DFA)*. It is well-known that for each NFA $\mathcal{A}$ one can construct a DFA $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and $|\mathcal{A}'| \in O(2^{|\mathcal{A}|})$.

Given the Büchi automaton $\mathcal{A}^{\varphi}_{\beta} = (Q_\beta, q_{I,\beta}, \Delta_\beta, F_\beta)$, we first compute the set $F^{\star}_{\beta} = \{q \in Q_\beta \mid \mathcal{L}(\mathcal{A}^{\varphi}_{\beta}(q)) \neq \emptyset\}$, where $\mathcal{A}^{\varphi}_{\beta}(q)$ denotes the Büchi automaton $\mathcal{A}^{\varphi}_{\beta}$ but with initial state q instead of q_I. Intuitively, the set $F^{\star}_{\beta}$ contains all states $q \in Q_\beta$ from which there exists an accepting run in $\mathcal{A}^{\varphi}_{\beta}$ and, hence, indicates whether a finite word $u \in \Sigma^*$ reaching a state of $F^{\star}_{\beta}$ can be extended to an infinite word $u\sigma' \in \mathcal{L}(\mathcal{A}^{\varphi}_{\beta})$. The set $F^{\star}_{\beta}$ can be computed, for instance, using a nested depth-first search [49] for each state $q \in Q_\beta$. Since each such search requires time quadratic in $|\mathcal{A}^{\varphi}_{\beta}|$, the set $F^{\star}_{\beta}$ can be computed in time $O(|\mathcal{A}^{\varphi}_{\beta}|^3)$.

Using $F^{\star}_{\beta}$, we define the NFA $\mathcal{B}^{\varphi}_{\beta} = (Q_\beta, q_{I,\beta}, \Delta_\beta, F^{\star}_{\beta})$. It shares the transition structure of $\mathcal{A}^{\varphi}_{\beta}$ and uses $F^{\star}_{\beta}$ as the set of final states. The next lemma states that $\mathcal{B}^{\varphi}_{\beta}$ indeed recognizes prefixes of words in $\mathcal{L}(\mathcal{A}^{\varphi}_{\beta})$.

Lemma 4.2. *Let* $\beta \in \mathbb{B}_4$ *and* $u \in \Sigma^*$. *Then,* $u \in \mathcal{L}(\mathcal{B}^{\varphi}_{\beta})$ *if and only if there exists an infinite word* $\sigma \in \Sigma^{\omega}$ *with* $V_r(u\sigma, \varphi) = \beta$.

Before we continue to the last step in our construction, let us briefly comment on the complexity of computing the NFAs $\mathcal{B}^{\varphi}_{\beta}$. Since $\mathcal{B}^{\varphi}_{\beta}$ and $\mathcal{A}^{\varphi}_{\beta}$ share the same underlying transition structure, we immediately obtain $|\mathcal{B}^{\varphi}_{\beta}| \in 2^{O(|\varphi|)}$. Moreover, the construction of $\mathcal{B}^{\varphi}_{\beta}$ is dominated by the computation of the set $F^{\star}_{\beta}$ and, hence, can be done in time $2^{O(|\varphi|)}$.

Step 3: In the final step, we construct a Moore machine implementing an rLTL monitor for φ. Formally, a *Moore machine* is a five-tuple $\mathcal{M} = (Q, q_I, \delta, \Gamma, \lambda)$ consisting of a deterministic transition structure (Q, q_I, δ), an output alphabet Γ, and an output function $\lambda \colon Q \to \Gamma$. The size of $\mathcal{M}$ as well of runs of $\mathcal{M}$ are defined as for DFAs. In contrast to a DFA, however, a Moore machine $\mathcal{M}$ computes a function $\lambda_{\mathcal{M}} \colon \Sigma^* \to \Gamma$ that is defined by $\lambda_{\mathcal{M}}(u) = \lambda(q_n)$ where q_n is the last state reached on the unique finite run $q_0 \cdots q_n$ of $\mathcal{M}$ on its input $u \in \Sigma^*$.

The first step in the construction of the Moore machine is to determinize the NFAs $\mathcal{B}^{\varphi}_{\beta}$, obtaining equivalent DFAs $C^{\varphi}_{\beta} = (Q'_\beta, q'_{I,\beta}, \delta'_\beta, F'_\beta)$ of at most exponential size in $|\mathcal{B}^{\varphi}_{\beta}|$. Subsequently, we combine these DFAs into a single Moore machine $\mathcal{M}_\varphi$ implementing the desired rLTL monitor. Intuitively, this Moore machine is the product of the DFAs C^{φ}_{β} for each $\beta \in \mathbb{B}_4$ and tracks the run of each individual DFA on the given input. Formally, $\mathcal{M}_\varphi$ is defined as follows.

Definition 4.3. Let $\mathbb{B}_4 = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$. We define $\mathcal{M}_\varphi = (Q, q_I, \Gamma, \delta, \lambda)$ by

- $Q = Q'_{\beta_1} \times Q'_{\beta_2} \times Q'_{\beta_3} \times Q'_{\beta_4} \times Q'_{\beta_5}$;
- $q_I = (q'_{I,\beta_1}, q'_{I,\beta_2}, q'_{I,\beta_3}, q'_{I,\beta_4}, q'_{I,\beta_5})$;
- $\delta\big((q_1, q_2, q_3, q_4, q_5), a\big) = (q'_1, q'_2, q'_3, q'_4, q'_5)$ where $q'_j = \delta'_{\beta_j}(q_j, a)$ for each $j \in \{1, \ldots, 5\}$;
- $\Gamma = \mathbb{B}^?_4$; and
- $\lambda\big((q_1, q_2, q_3, q_4, q_5)\big) = \xi\big(\{\beta_j \in \mathbb{B}_4 \mid q_j \in F'_{\beta_j}, j \in \{1, \ldots, 5\}\}\big)$,

where the surjective function $\xi \colon 2^{\mathbb{B}_4} \to \mathbb{B}^?_4$ translates sets $B \subseteq \mathbb{B}_4$ of truth values to the robust monitoring semantics as follows: $\xi(B) = \beta^? \in \mathbb{B}^?_4$ with

$$\beta^?[j] = \begin{cases} 0 & \text{if } \beta[j] = 0 \text{ for each } \beta \in B; \\ 1 & \text{if } \beta[j] = 1 \text{ for each } \beta \in B; \text{ and} \\ ? & \text{otherwise.} \end{cases}$$

The main result of this paper now shows that the Moore machine $\mathcal{M}_\varphi$ implements V^m_r, i.e., we have $\lambda_{\mathcal{M}_\varphi}(u) = V^m_r(u, \varphi)$ for every prefix u.

Theorem 4.4. *For every rLTL formula* φ, *one can construct an rLTL monitor of size* $2^{2^{O(|\varphi|)}}$.

In a final post-processing step, we minimize $\mathcal{M}_\varphi$ (e.g., using one of the standard algorithms for deterministic automata). As a result, we obtain the unique minimal monitor for the given rLTL formula.

It is left to determine the complexity of our rLTL monitor construction. Since each DFA C^{φ}_{β} is in the worst case exponential in the size of the NFA $\mathcal{B}^{\varphi}_{\beta}$, we immediately obtain that C^{φ}_{β} is at most of size $2^{2^{O(|\varphi|)}}$. Thus, the Moore machine $\mathcal{M}_\varphi$ is at most of size $2^{2^{O(|\varphi|)}}$ as well and can be effectively computed in doubly-exponential time in $|\varphi|$. Note that this matches the complexity bound of Bauer et al.'s approach for LTL runtime monitoring [13]. Moreover, this bound is tight since rLTL subsumes LTL and a doubly-exponential bound is tight for LTL [13, 36]. Hence, robust runtime monitoring asymptotically incurs no extra cost compared to classical LTL runtime monitoring. However, it provides more useful information as we demonstrate next in our experimental evaluation.

5 EXPERIMENTAL EVALUATION

Besides incorporating a notion of robustness into classical LTL monitoring, our rLTL monitoring approach also promises to provide richer information than its LTL counterpart. In this section, we evaluate empirically whether this promise is actually fulfilled. More precisely, we answer the following two questions on a comprehensive suite of benchmarks:

(1) How does rLTL monitoring compare to classical LTL monitoring in terms of monitorability?
(2) For formulas that are both LTL-monitorable and rLTL-monitorable, how do both approaches compare in terms of the size of the resulting monitors and the time required to construct them?

To answer these research questions, we have implemented a prototype, which we named `rLTL-mon`. Our prototype is written in Java and builds on top of two libraries: Owl [35], a library for LTL and automata over infinite words, as well as AutomataLib (part of LearnLib [32]), a library for automata over finite words and Moore machines. For technical reasons (partly due to limitations of the Owl library and partly to simplify the implementation), `rLTL-mon` uses a monitor construction that is slightly different from the one described in Section 4: Instead of translating an rLTL formula into nondeterministic Büchi automata, `rLTL-mon` constructs deterministic parity automata. These parity automata are then directly converted into DFAs, thus skirting the need for a detour over NFAs and a subsequent determinization step. Note, however, that this alternative construction produces the same rLTL monitors than the one described in Section 4. Moreover, it has the same asymptotic complexity. The (anonymized) sources of our prototype are available online under the MIT license.[8]

Benchmarks and Experimental Setup. Starting point of our evaluation was the original benchmark suite of Bauer et al. [13], which is based on a survey by Dwyer on frequently used software specification patterns [20]. This benchmark suite consists of 97 LTL formulas and covers a wide range of patterns, including safety, scoping, precedence, and response patterns. For our rLTL monitor construction, we interpreted each LTL formula in the benchmark suite as an rLTL formula (by treating every operator as a robust operator).

We compared `rLTL-mon` to Bauer et al.'s implementation of their LTL monitoring approach, which the authors named `LTL3 tools`. This tool uses LTL2BA [28] to translate LTL formulas into Büchi automata and AT&T's fsmlib as a means to manipulate finite-state machines. Since LTL2BA's and Owl's input format for LTL formulas do not match exactly, we have translated all benchmarks into a suitable format using a python script.

We conducted all experiments on an Intel Core i5-6600 @ 3.3 GHz in a virtual machine with 4 GB of RAM running Ubuntu 18.04 LTS. As no monitor construction took longer than 600 *s*, we did not impose any time limit.

Results. Our evaluation shows that `LTL3 tools` and `rLTL-mon` are both able to generate monitors for all 97 formulas in Bauer et al.'s benchmark suite.[9] Aggregated statistics of this evaluation are visualized in Figure 1.[10]

The histogram in Figure 1a shows the aggregate number of LTL and rLTL monitors with respect to their number of states. As Bauer et al. already noted in their original work, the resulting LTL monitors are quite small (none had more than six states), which they attribute to Dwyer et al.'s specific selection of formulas [20]. A similar observation is also true for the rLTL monitors: None had more than eight states.

To determine which formulas are monitorable and which are not, we used a different, though equivalent definition, which is easy to check on the monitor itself: an LTL formula (rLTL formula) is monitorable if and only if the unique minimized LTL monitor (rLTL monitor) does not contain a sink-state with universal self-loop that outputs "?" (that outputs "????"). Bauer et al. report that 44.3 % of all LTL monitors (43 out of 97) have this property (in fact, exactly the 43 LTL monitors with a single state), which means that 44.3 % of all formulas in their benchmark suite are not LTL-monitorable. By contrast, all these formulas are rLTL-monitorable. Moreover, in 78.4 % of the cases (76 out of 97), the rLTL monitor has more distinct outputs than the LTL monitor, indicating that the rLTL monitor provides more fine-grained information of the property being monitored; in the remaining 21.6 %, both monitors have the same number of distinct outputs. These results answer our first research question strongly in favor of rLTL monitoring: *rLTL monitoring did in fact provide more information than its classical LTL counterpart. In particular, only* 55.7 % *of the benchmarks are LTL-monitorable, whereas* 100 % *are rLTL-monitorable.*

Let us now turn to our second research question and compare both monitoring approaches on the 54 formulas that are both LTL-monitorable and rLTL-monitorable. For these formulas, Figure 1b further provides statistical analysis of the generated monitors in terms of their size (left diagram) as well as the time required to generate them (right diagram). Each box in the diagrams shows the lower and upper quartile (left and right border of the box, respectively), the median (line within the box), and minimum and maximum (left and right whisker, respectively).

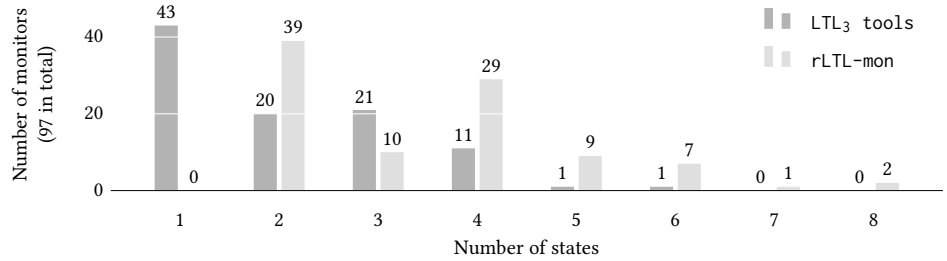
Let us first consider the size of the monitors (left diagram of Figure 1b). The majority of LTL monitors (52) has between two and four states, while the majority of rLTL monitors (45) has between two and five states. For 21 benchmarks, the LTL and rLTL monitors are of equal size, while the rLTL monitor is larger for the remaining 33 benchmarks (in no case is the LTL monitor larger than the rLTL monitor). On average, rLTL monitors are about 1.5 times larger than the corresponding LTL monitors.

Let us now discuss the time taken to construct the monitors. As the diagram on the right-hand-side of Figure 1b shows, `LTL3 tools` was considerably faster than `rLTL-mon` on a majority of benchmarks (around 0.1 *s* and 2.6 *s* per benchmark, respectively). For all 54 benchmarks, the rLTL monitor construction took longer than the construction of the corresponding LTL monitor (although there are two non-LTL-monitorable formulas for which the construction of the rLTL monitor was faster). However, we attribute this large

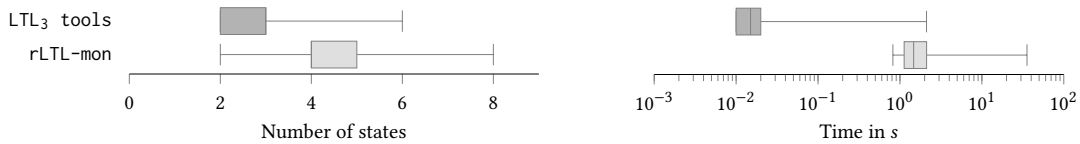
[8] https://gitlab.mpi-sws.org/rltl/rltl-runtime-monitoring-code

[9] Note that the tools disagreed on one monitor where `LTL3 tools` constructed a monitor with 1 state whereas `rLTL-mon` constructed an LTL monitor with 8 states. The respective formula was removed from the reported results.

[10] Detailed statistics can be found in the technical report.



(a) Histogram of the number of monitors with respect to their size



(b) Analysis of the monitor construction for the 54 formulas that are both LTL-monitorable and rLTL-monitorable

Figure 1: Comparison of `rLTL-mon` and `LTL3 tools` on Bauer et al.'s benchmarks [13]

runtime gap partly to the overhead caused by repeatedly starting the Java virtual machine, which is not required in the case of `LTL3 tools`. Note that this is not a concern in practice as a monitor is only constructed once before it is deployed.

Finally, our analysis answers our second question: *rLTL monitors are only slightly larger than the corresponding LTL monitors and although they require considerably more time to construct, the overall construction time was negligible for almost all benchmarks.*

6 CONCLUSION

We adapted the three-valued LTL monitoring semantics of Bauer et al. to rLTL, proved that the construction of monitors is asymptotically no more expensive than the one for LTL, and validated our approach on the benchmark of Bauer et al.: All formulas are rLTL-monitorable and the rLTL monitor is strictly more informative than its LTL counterpart for 77% of their formulas.

Recall Theorem 3.3, which states that the truth values 0011 and 0001 are not realizable. This points to a drawback regarding the two middle bits: When considering the formula $\boxdot a$, the second bit represents $\Diamond\Box a$ and the third bit $\Box\Diamond a$. A prefix cannot possibly provide enough information to distinguish these two formulas. On the other hand, the truth value ??11 is realizable, which shows that the middle bits can be relevant. In further work, we will investigate the role of the middle bits in rLTL monitoring.

Moreover, the informedness of a monitor can be increased further when attributing a special role to the last position(s) of a prefix. Even though a prefix of the form $\emptyset^+\{a\}^+$ does not fully satisfy $\Diamond\Box a$, neither does it fully violate it. If the system just now reached a state in which $\{a\}$ always holds, an infinite continuation of the execution would satisfy the specification. So rather than reporting an undetermined result, the monitor could indicate that an infinite repetition of the last position of the prefix would satisfy the specification. Similarly, for a prefix $\{a\}^+\emptyset$, the specification $\Box\Diamond a$ is undetermined. While an infinite repetition of the last position ($\{a\}^+\emptyset^\omega$) does not satisfy the specification, an infinite repetition of the last two positions ($(\{a\}^+(\emptyset\{a\})^\omega$) would. We plan to investigate an extension of rLTL which takes this observation into account.

Bauer et al. [11] proposed an orthogonal approach with the logic RV-LTL. Here, the specification can contain the strong (weak) next-operator whose operand is consider violated (satisfied) at the last position of the trace. A formula that is undetermined under the strong semantics and satisfied (violated) under the weak semantics is considered *potentially true (potentially false)*. Incorporating one of these approaches into rLTL monitoring could refine its output and thus increase its level of informedness.

Moreover, desired properties for cyber-physical systems often include real-time components such as "touch the ground at most 15 seconds after receiving a landing command". Monitors for logics taking real-time into account [14], such as STL [39, 40], induce high computational overhead at runtime when compared to LTL and rLTL monitors. Thus, a real-time extension for rLTL retaining its low runtime cost would greatly increase its viability as specification language.

ACKNOWLEDGMENTS

The authors would like to thank Li Bingchen for discovering the formula mentioned in Footnote 6. The work of Maximilian Schwenger was supported by the European Research Council (ERC) Grant OSARES (No. 683300) and the German Research Foundation (DFG) as part of the Collaborative Research Center "Center for Perspicuous Computing" (TRR 248, 389792660). The work of Paulo Tabuada was partially supported by the NSF project 1645824. The work of Alexander Weinert was supported by the Saarbrücken Graduate School of Computer Science. The work of Martin Zimmermann was supported by the Engineering and Physical Sciences Research Council (EPSRC) EP/S032207/1.

REFERENCES

[1] Houssam Abbas, Alena Rodionova, Ezio Bartocci, Scott A. Smolka, and Radu Grosu. Quantitative regular expressions for arrhythmia detection algorithms. In Jérôme Feret and Heinz Koeppl, editors, *CMSB 2017*, volume 10545 of *LNCS*, pages 23–39. Springer, 2017. doi:10.1007/978-3-319-67471-1_2.

[2] Florian-Michael Adolf, Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Christoph Torens. Stream runtime monitoring on UAS. In Shuvendu K. Lahiri and Giles Reger, editors, *RV 2017*, volume 10548 of *LNCS*, pages 33–49. Springer, 2017. doi:10.1007/978-3-319-67531-2_3.

[3] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015*, volume 9207 of *LNCS*, pages 356–374. Springer, 2015. doi:10.1007/978-3-319-21668-3_21.

[4] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. Regular programming for quantitative properties of data streams. In Peter Thiemann, editor, *ESOP 2016*, volume 9632 of *LNCS*, pages 15–40. Springer, 2016. doi:10.1007/978-3-662-49498-1_2.

[5] Tzanis Anevlavis, Daniel Neider, Matthew Phillipe, and Paulo Tabuada. Evrostos: the rLTL verifier. In Necmiye Ozay and Pavithra Prabhakar, editors, *HSCC 2019*, pages 218–223. ACM, 2019. doi:10.1145/3302504.3311812.

[6] Tzanis Anevlavis, Matthew Philippe, Daniel Neider, and Paulo Tabuada. Verifying rLTL formulas: now faster than ever before! In *CDC 2018*, pages 1556–1561. IEEE, 2018. doi:10.1109/CDC.2018.8619014.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[8] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012*, volume 7436 of *LNCS*, pages 68–84. Springer, 2012. doi:10.1007/978-3-642-32759-9_9.

[9] Ezio Bartocci, Roderick Bloem, Dejan Nickovic, and Franz Röck. A counting semantics for monitoring LTL specifications over finite traces. In Hana Chockler and Georg Weissenbacher, editors, *CAV 2018*, volume 10981 of *LNCS*, pages 547–564. Springer, 2018. doi:10.1007/978-3-319-96145-3_29.

[10] David A. Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zalinescu. Monitoring of temporal first-order properties with aggregations. *Formal Methods in System Design*, 46(3):262–285, 2015. doi:10.1007/s10703-015-0222-7.

[11] Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Tasiran, editors, *RV 2007*, volume 4839 of *LNCS*, pages 126–138. Springer, 2007. doi:10.1007/978-3-540-77395-5_11.

[12] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *J. Log. Comput.*, 20(3):651–674, 2010. doi:10.1093/logcom/exn075.

[13] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011. doi:10.1145/2000799.2000800.

[14] Arthur J. Bernstein and Paul K. Harter Jr. Proving real-time properties of programs with temporal logic. In John Howard and David P. Reed, editors, *SOSP 1981*, pages 1–11. ACM, 1981. doi:10.1145/800216.806585.

[15] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. Lustre: A declarative language for programming synchronous systems. In *POPL 1987*, pages 178–188. ACM Press, 1987. doi:10.1145/41625.41641.

[16] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *TIME 2005*, pages 166–174. IEEE Computer Society, 2005. doi:10.1109/TIME.2005.26.

[17] Normann Decker, Martin Leucker, and Daniel Thoma. Impartiality and anticipation for monitoring of visibly context-free properties. In Axel Legay and Saddek Bensalem, editors, *RV 2013*, volume 8174 of *LNCS*, pages 183–200. Springer, 2013. doi:10.1007/978-3-642-40787-1_11.

[18] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 264–279. Springer, 2013. doi:10.1007/978-3-642-39799-8_19.

[19] Doron Drusinsky. The temporal rover and the ATG rover. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN 2000*, volume 1885 of *LNCS*, pages 323–330. Springer, 2000. doi:10.1007/10722468_19.

[20] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *ICSE 1999*, pages 411–420. ACM, 1999. doi:10.1145/302405.302672.

[21] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In Warren A. Hunt and Fabio Somenzi, editors, *CAV 2003*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003. doi:10.1007/978-3-540-45069-6_3.

[22] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262 – 4291, 2009. URL: http://www.sciencedirect.com/science/article/pii/S0304397509004149, doi:https://doi.org/10.1016/j.tcs.2009.06.021.

[23] Yliès Falcone and César Sánchez, editors. *RV 2016*, volume 10012 of *LNCS*. Springer, 2016. doi:10.1007/978-3-319-46982-9.

[24] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, L. Tentrup, M. Stenger, and H. Torfah. Streamlab: Stream-based monitoring of cyber-physical systems. In *CAV 2019*, 2019. To appear.

[25] Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Hazem Torfah. A stream-based specification language for network monitoring. In Falcone and Sánchez [23], pages 152–168. doi:10.1007/978-3-319-46982-9_10.

[26] Bernd Finkbeiner, Sriram Sankaranarayanan, and Henny Sipma. Collecting statistics over runtime executions. *Form. Meth. in Sys. Des.*, 27(3):253–274, 2005. doi:10.1007/s10703-005-3399-3.

[27] Bernd Finkbeiner and Hazem Torfah. The density of linear-time properties. In Deepak D'Souza and K. Narayan Kumar, editors, *ATVA 2017*, volume 10482 of *LNCS*, pages 139–155. Springer, 2017. doi:10.1007/978-3-319-68167-2_10.

[28] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV 2001*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001. doi:10.1007/3-540-44585-4_6.

[29] Sylvain Hallé. When RV meets CEP. In Falcone and Sánchez [23], pages 68–91. doi:10.1007/978-3-319-46982-9_6.

[30] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS 2002*, volume 2280 of *LNCS*, pages 342–356. Springer, 2002. doi:10.1007/3-540-46002-0_24.

[31] Klaus Havelund and Grigore Rosu. An overview of the runtime verification tool Java PathExplorer. *Form. Meth. in Sys. Des.*, 24(2):189–215, 2004. doi:10.1023/B:FORM.0000017721.39909.4b.

[32] Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015 (Part I)*, volume 9206 of *LNCS*, pages 487–495. Springer, 2015. doi:10.1007/978-3-319-21690-4_32.

[33] Stefan Jaksic, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Nickovic. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1):83–112, 2018. doi:10.1007/s10703-018-0319-x.

[34] Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. LTL store: Repository of LTL formulae from literature and case studies. *arXiv*, 1807.03296, 2018. URL: http://arxiv.org/abs/1807.03296, arXiv:1807.03296.

[35] Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In Lahiri and Wang [37], pages 543–550. doi:10.1007/978-3-030-01090-4_34.

[36] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Form. Meth. in Sys. Des.*, 19(3):291–314, 2001. doi:10.1023/A:1011254632723.

[37] Shuvendu K. Lahiri and Chao Wang, editors. *ATVA 2018*, volume 11138 of *LNCS*. Springer, 2018. doi:10.1007/978-3-030-01090-4.

[38] Insup Lee, Sampath Kannan, Moonjoo Kim, Oleg Sokolsky, and Mahesh Viswanathan. Runtime assurance based on formal specifications. In Hamid R. Arabnia, editor, *PDPTA 1999*, pages 279–287. CSREA Press, 1999.

[39] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *FORMATS and FTRTFT 2004*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

[40] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *LNCS*, pages 475–505. Springer, 2008. doi:10.1007/978-3-540-78127-1_26.

[41] Oded Maler and Amir Pnueli. Timing analysis of asynchronous circuits using timed automata. In Paolo Camurati and Hans Eveking, editors, *CHARME 1995*, volume 987 of *LNCS*, pages 189–205. Springer, 1995. doi:10.1007/3-540-60385-9_12.

[42] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

[43] Ramy Medhat, Borzoo Bonakdarpour, Sebastian Fischmeister, and Yogi Joshi. Accelerated runtime verification of LTL specifications with counting semantics. In Falcone and Sánchez [23], pages 251–267. doi:10.1007/978-3-319-46982-9_16.

[44] Patrick Moosbrugger, Kristin Y. Rozier, and Johann Schumann. R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Form. Meth. in Sys. Des.*, 51(1):31–61, 2017. doi:10.1007/s10703-017-0275-x.

[45] Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. Copilot: A hard real-time runtime monitor. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *RV 2010*, volume 6418 of *LNCS*, pages 345–359. Springer, 2010. doi:10.1007/978-3-642-16612-9_26.

[46] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006*, volume 4085 of *LNCS*, pages 573–586. Springer, 2006. doi:10.1007/11813040_38.

[47] Alena Rodionova, Ezio Bartocci, Dejan Nickovic, and Radu Grosu. Temporal logic as filtering. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, pages 11–20, New York, NY, USA, 2016. ACM. URL: http://doi.acm.org/10.1145/2883817.2883839, doi:10.1145/2883817.2883839.

[48] Martin Roesch. Snort: Lightweight intrusion detection for networks. In David W. Parter, editor, *LISA 1999*, pages 229–238. USENIX, 1999.

[49] Stefan Schwoon and Javier Esparza. A note on on-the-fly verification algorithms. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS 2005*, volume 3440 of *LNCS*, pages 174–190. Springer, 2005. doi:10.1007/978-3-540-31980-1_12.

[50] Simone Silvetti, Laura Nenzi, Ezio Bartocci, and Luca Bortolussi. Signal convolution logic. In Lahiri and Wang [37], pages 267–283. doi:10.1007/978-3-030-01090-4_16.

[51] Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In Jean-Marc Talbot and Laurent Regnier, editors, *CSL 2016*, volume 62 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - LZI, 2016. doi:10.4230/LIPIcs.CSL.2016.10.

[52] Hazem Torfah and Martin Zimmermann. The complexity of counting models of linear-time temporal logic. *Acta Inf.*, 55(3):191–212, 2018. doi:10.1007/s00236-016-0284-z.

[53] Xian Zhang, Martin Leucker, and Wei Dong. Runtime verification with predictive semantics. In Alwyn Goodloe and Suzette Person, editors, *NFM 2012*, volume 7226 of *LNCS*, pages 418–432. Springer, 2012. doi:10.1007/978-3-642-28891-3_37.