Comparing LTL Semantics for Runtime Verification

ANDREAS BAUER, *National ICT Australia (NICTA) & Australian National University, Canberra, Australia. E-mail: andreas.bauer@nicta.com.au*

MARTIN LEUCKER, *Institut für Informatik, Technische Universität München, München, Germany. E-mail: leucker@in.tum.de*

CHRISTIAN SCHALLHART, *Fachbereich Informatik, Technische Universität Darmstadt, Darmstadt, Germany. E-mail: schallhart@forsyte.de*

Abstract

When monitoring a system w.r.t. a property defined in a temporal logic such as LTL, a major concern is to settle with an adequate interpretation of observable system events; that is, models of temporal logic formulae are usually infinite words of events, whereas at runtime only finite but incrementally expanding prefixes are available.

In this work, we review LTL-derived logics for finite traces from a runtime-verification perspective. In doing so, we establish four maxims to be satisfied by any LTL-derived logic aimed at runtime verification. As no pre-existing logic readily satisfies all of them, we introduce a new four-valued logic Runtime Verification Linear Temporal Logic RV-LTL in accordance to these maxims. The semantics of Runtime Verification Linear Temporal Logic (RV-LTL) indicates whether a finite word describes a system behaviour which either (i) satisfies the monitored property, (ii) violates the property, (iii) will presumably violate the property, or (iv) will presumably conform to the property in the future, once the system has stabilized. Notably, (i) and (ii) correspond to the classical semantics of LTL, whereas (iii) and (iv) are chosen whenever an observed system behaviour has not yet lead to a violation or acceptance of the monitored property.

Moreover, we present a monitor construction for RV-LTL properties in terms of Moore machines signalizing the semantics of the so far obtained execution trace w.r.t. the monitored property.

Keywords: Runtime verification, temporal logic, monitoring

1 Introduction

Runtime verification of a given correctness property φ formulated in linear temporal logic (LTL) [18] requires at its core the evaluation of the semantics of φ w.r.t. to a finite observed system behaviour. But the evaluation of LTL properties on finite traces proved to be an obstacle, as LTL is usually evaluated over infinite traces and since the standard semantics of LTL on finite traces [15] is unsatisfactory for the purpose at hand.

While the syntax and semantics of LTL on infinite traces is well accepted in the literature, there is no consensus on defining LTL over finite traces. Besides the definition in [15], a number of *two-valued* semantics for LTL on finite traces have been proposed [9, 13, 14, 12, 20, 6], see Eisner *et al.* [8] for a comprehensive survey on this topic. Alternatively, it has been proposed to restrict the syntax of LTL for runtime verification, such that formulae which may contain certain future obligations cannot be specified at all [10].

In monitoring a property, there arise at least three different situations: in the first case, the property is satisfied after a finite number of steps, independently of the future continuation; second, the property is shown to evaluate to false for every possible continuation, and third, the finite, already observed prefix still allows different continuations leading to either satisfaction or falsification. A prefix leading

Vol. 20 No. 3, © The Author, 2010. Published by Oxford University Press. All rights reserved.
For Permissions, please email: journals.permissions@oxfordjournals.org
Published online 4 February 2010 doi:10.1093/logcom/exn075

to the first (second) case is called a good (bad) prefix [16]. Thus, every two-valued logic must evaluate to true or false prematurely since it cannot reflect the third and inconclusive case properly.

To overcome these obstacles, we propose in Refs [1, 2] the three-valued logic LTL_3 over finite traces. There, a property evaluates to *true (false)*, w.r.t. a finite word, whenever the observed word is a good (bad) prefix. In all other cases, the word is evaluated to an *inconclusive* verdict.

This scheme matches well with the notion of *safety* (e.g. Gp—always p) and *co-safety* (e.g. Fp—finally p) properties, since these are either finitely refutable or satisfiable. The union of safety and co-safety properties forms a strict subset of the *monitorable* properties, as defined and shown in Ref. [3]: a property is monitorable for some prefix, as long as there is some continuation leading to a good or bad prefix, i.e. as long as it is possible to arrive at a definite true or false verdict.

However, there remain many properties which are *non-monitorable:* consider for example the request/acknowledge property $G(r \to Fa)$ which states that every request is finally acknowledged. No finite word is a good or bad prefix for $G(r \to Fa)$ and therefore, this property is always evaluated to an inconclusive verdict. But since such properties arise often in practice, such a solution is quite *ugly*—raising the question of whether it is possible to refine the inconclusive verdict into a more telling verdict. For the request/acknowledge property, for example, we aim to have a semantics such that

– a finite string ending in a (i.e. all requests have been acknowledged) yields a truth value which indicates that φ is probably satisfied; whereas,
– a finite string ending in r (i.e. there is a request not acknowledged yet) should evaluate to a truth value which expresses that φ is likely to remain unsatisfied.

In this work, we examine how to determine a more detailed evaluation of such an inconclusive verdict. To this end, we recall in Section 3 three pre-existing logics which are based upon LTL and which feature a semantics on finite words, namely FLTL [17], $\text{LTL}^{\mp}$ [8], as well as LTL_3 [2]. Since we are comparing these logics, we present them in a unified syntactical framework, outlined together with standard LTL in Section 2.

Next we discuss in Section 4 the usefulness of these logics for the purpose of runtime verification. To this end, we define four maxims that we consider relevant for a linear temporal logic when using it for runtime verification.

Since none of the pre-existing logics satisfies all of the four maxims, we introduce in Section 5 a four-valued semantics for LTL which refines the inconclusive verdict into a *presumably true* and *presumably false* truth value. We call the resulting logic *Runtime Verification Linear Temporal Logic* (RV-LTL). We show that RV-LTL's semantics indeed adheres to our four maxims and that it matches our intuition for request/response properties. RV-LTL seems to correspond to the semantics realized by the Temporal Rover [7] and has, to the best of our knowledge, not been formally captured elsewhere.

Finally, we define in Section 6 a translation to generate a monitor procedure in terms of a Moore machine of minimal size for each given property φ. Such a monitor implements the RV-LTL semantics and computes a new verdict with each upcoming state on the incrementally expanding system trace.

This article extends [4] by giving a formal treatment of the four maxims and a comprehensive comparison of LTL derived logics on finite traces, from a runtime verification perspective.

2 Preliminaries

2.1 Truth domains

We consider in this article the traditional two-valued semantics with truth values *true*, denoted with $\top$, and *false*, denoted with $\bot$, next to truth values that give more information to which degree a formula is considered satisfied or not. Since truth values should be comparable and combinable in terms of Boolean operations expressed by the connectives of the underlying logic, we interpret these truth values as elements of some lattice.

A *lattice* is a partially ordered set $(\mathcal{L},\sqsubseteq)$ where for each $x,y\in\mathcal{L}$, there exists (i) a unique *greatest lower bound* (glb), which is called the *meet* of x and y, and is denoted with $x\sqcap y$, and (ii) a unique *least upper bound* (lub), which is called the *join* of x and y, and is denoted with $x\sqcup y$. A lattice is called *finite* iff $\mathcal{L}$ is finite. Every finite lattice has a well-defined unique least element, called *bottom*, denoted with $\bot$, and analogously a greatest element, called *top*, denoted with $\top$. A lattice is *distributive*, iff $x\sqcap(y\sqcup z)=(x\sqcap y)\sqcup(x\sqcap z)$, and, dually, $x\sqcup(y\sqcap z)=(x\sqcup y)\sqcap(x\sqcup z)$. In a *de Morgan* lattice, every element x has a unique *dual* element $\overline{x}$, such that $\overline{\overline{x}}=x$ and $x\sqsubseteq y$ implies $\overline{y}\sqsubseteq\overline{x}$. A distributive lattice is called *Boolean* iff $x\sqcup\overline{x}=\top$ and $x\sqcap\overline{x}=\bot$. As the common denominator of the semantics for the subsequently defined logics is a finite de Morgan lattice, we take this to our understanding of a truth domain.

Definition 1 (Truth domain)
We call $\mathcal{L}$ a *truth domain*, if it is a finite de Morgan lattice.

The two valued truth domain $\mathbb{B}_2=\{\bot,\top\}$ is a Boolean lattice with $\bot\sqsubseteq\top$ and $\sqcap$ and $\sqcup$ defined in the expected manner.

2.2 *LTL on infinite traces*

As a starting point for all subsequently defined logics, we first recall LTL interpreted over infinite traces, as introduced by Pnueli [18] in the setting of specification and verification.

In case of LTL over infinite traces, one is used to have a syntax ranging over a small set of temporal and Boolean operators and to add additional operators by means of abbreviations. For example, the conjunction is often expressed indirectly using negation and disjunction—at least in the two valued truth domain. However, as several of the equivalences known from LTL over infinite traces do not hold in all the logics over finite traces considered in the article, we start with a comprehensive—and with respect to standard LTL—redundant set of Boolean and temporal operators. We only deviate from this approach for *finally* (F) and *globally* (G) operators, as well as for implication ($\rightarrow$).

For the remainder of this article, let AP be a finite and non-empty set of *atomic propositions* and $\Sigma=2^{\mathrm{AP}}$ a finite *alphabet*. We write a_i for any single element of Σ, i.e. a_i is a possibly empty subset of propositions taken from AP.

Finite traces (which we call interchangeably words) over Σ are elements of Σ^*, usually denoted with $u,u',u_1,u_2,\ldots$. The empty trace is denoted with ϵ. Infinite traces are elements of Σ^ω, usually denoted with $w,w',w_1,w_2,\ldots$ For some infinite trace $w=a_0a_1\ldots$, we denote with w^i the suffix $a_ia_{i+1}\ldots$. In case of a finite trace $u=a_0a_1\ldots a_{n-1}$, u^i denotes the suffix $a_ia_{i+1}\ldots a_{n-1}$ for $0\leq i<n$ and the empty string ϵ for $n\leq i$.

The set of LTL formulae is defined using *true*, the atomic propositions $p\in\mathrm{AP}$, *disjunction*, *next* X and *until* U, as positive operators, together with *negation* $\neg$. For comparison with logics over finite traces, we moreover add dual operators, namely *false*, $\neg p$, *weak next* $\bar{X}$ and *release* R, respectively.

Definition 2 (Syntax of LTL formulae)
Let p be an atomic proposition from a finite set of atomic propositions AP. The set of LTL formulae, denoted with LTL, is inductively defined by the following grammar:

$$\begin{aligned}
\varphi &::= true \mid p \mid \varphi\vee\varphi \mid \varphi\,U\,\varphi \mid X\varphi\\
\varphi &::= false \mid \neg p \mid \varphi\wedge\varphi \mid \varphi R\varphi \mid \bar{X}\varphi\\
\varphi &::= \neg\varphi
\end{aligned}$$

Boolean constants

$$\begin{aligned}[w \models \mathit{true}]_\omega &= \top \\ [w \models \mathit{false}]_\omega &= \bot\end{aligned}$$

Boolean combinations

$$\begin{aligned}[w \models \neg\varphi]_\omega &= \overline{[w \models \varphi]_\omega} \\ [w \models \varphi \vee \psi]_\omega &= [w \models \varphi]_\omega \sqcup [w \models \psi]_\omega \\ [w \models \varphi \wedge \psi]_\omega &= [w \models \varphi]_\omega \sqcap [w \models \psi]_\omega\end{aligned}$$

atomic propositions

$$\begin{aligned}[w \models p]_\omega &= \begin{cases}\top & \text{if } p \in a_0 \\ \bot & \text{if } p \notin a_0\end{cases} \\ [w \models \neg p]_\omega &= \begin{cases}\top & \text{if } p \notin a_0 \\ \bot & \text{if } p \in a_0\end{cases}\end{aligned}$$

(weak) next

$$\begin{aligned}[w \models X\varphi]_\omega &= [w^1 \models \varphi]_\omega \\ [w \models \bar{X}\varphi]_\omega &= [w^1 \models \varphi]_\omega\end{aligned}$$

until/release

$$[w \models \varphi\ U\ \psi]_\omega = \begin{cases}\top & \text{there is a } k \geq 0 : [w^k \models \psi]_\omega = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \leq l < k : [w^l \models \varphi] = \top \\ \bot & \text{else}\end{cases}$$

$$[w \models \varphi\ R\ \psi]_\omega = \begin{cases}\top & \text{for all } k \geq 0 : [w^k \models \psi]_\omega = \top \text{ or} \\ & \text{there is a } k \geq 0 : [w^k \models \varphi]_\omega = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \leq l \leq k : [w^l \models \psi] = \top \\ \bot & \text{else}\end{cases}$$

FIGURE 1. Semantics of LTL formulae over an infinite traces $w = a_0a_1 \ldots \in \Sigma^\omega$

Moreover, we define by means of abbreviation the *finally* F and *globally* G operators as

$$F\varphi := \mathit{true}\, U\, \varphi \quad \text{and} \quad G\varphi := \neg F \neg\varphi$$

as well as *implication* $\varphi \rightarrow \psi$ as a shorthand for $\neg\varphi \vee \psi$.

LTL formulae over infinite traces are interpreted as usual over the two valued truth domain $\mathbb{B}_2$.

DEFINITION 3 (Semantics of LTL)
The semantics of LTL formulae over infinite traces $w = a_0a_1 \ldots \in \Sigma^\omega$ is given by the function $[\_ \models \_]_\omega : \Sigma^\omega \times \text{LTL} \rightarrow \mathbb{B}_2$, which is defined inductively as shown in in Figure 1.

Inspecting the semantics, we observe that there is no difference of X and $\bar{X}$ in LTL over infinite traces. However, $\bar{X}$ acts differently when finite words are considered. Moreover, the semantics for a negated atomic proposition $\neg p$ is actually given twice: once explicitly and once inductively via negation. Fortunately, both definitions coincide, ensuring that the semantics of LTL is well-defined.

We call $w \in \Sigma^\omega$ a *model* of φ iff $[w \models \varphi] = \top$. For every LTL formula φ, its set of models, denoted with $\mathcal{L}(\varphi)$, is a regular set of infinite traces which is accepted by a corresponding Büchi automaton [22, 21].

In the next section, we introduce several versions of LTLs for finite traces and compare these logics by means of certain properties, such as the induced equivalences on formulae. To this end, we consider LTLs $\mathfrak{L}$ with a syntax as in Definition 2, together with a semantic function $[\_ \models \_]_{\mathfrak{L}} : \Sigma^{\omega/*} \times \text{LTL} \rightarrow \mathbb{B}_{\mathcal{L}}$ that yields an element of the truth domain $\mathbb{B}_{\mathcal{L}}$, given an infinite or finite trace and an LTL formula. Then, for two formulae $\varphi, \psi \in \text{LTL}$, we say that φ is equivalent to ψ, denoted with $\varphi \equiv_{\mathfrak{L}} \psi$, iff for all $w \in \Sigma^{\omega/*}$, we have

$$[w \models \varphi]_{\mathfrak{L}} = [w \models \psi]_{\mathfrak{L}}$$

Tertium-non-datur laws

$$\varphi \vee \neg\varphi \equiv \mathit{true}$$
$$\varphi \wedge \neg\varphi \equiv \mathit{false}$$

distributive laws

$$\varphi \vee (\psi \wedge \eta) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \eta)$$
$$\varphi \wedge (\psi \vee \eta) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \eta)$$

de Morgan laws

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$
$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$
$$\neg\neg\varphi \equiv \varphi$$

de Morgan-X law

$$\neg X\varphi \equiv_\omega \bar{X}\neg\varphi$$

de Morgan-U/R laws

$$\neg(\varphi\ U\ \psi) \equiv \neg\varphi\ R\ \neg\psi$$
$$\neg(\varphi\ R\ \psi) \equiv \neg\varphi\ U\ \neg\psi$$

unwinding laws

$$\varphi\ U\ \psi \equiv \psi \vee (\varphi \wedge X(\varphi\ U\ \psi))$$
$$\varphi\ R\ \psi \equiv \psi \wedge (\varphi \vee \bar{X}(\varphi\ R\ \psi))$$

FIGURE 2. Fundamental equivalence laws

In particular, for LTL, we denote this equivalence by $\equiv_\omega$. We say that logic $\mathfrak{L}$ satisfies the *Boolean laws* if for all traces w and all formulae φ

$$[w \models \varphi \vee \neg\varphi]_{\mathfrak{L}} = \top \text{ and } [w \models \varphi \wedge \neg\varphi]_{\mathfrak{L}} = \bot$$

REMARK 1
LTL satisfies the Boolean laws.

We say that the logic $\mathfrak{L}$ satisfies a certain *equivalence law*, if its equivalences given in Figure 2 hold. We denote the equivalence laws shown there as the *fundamental equivalence laws.*

REMARK 2
LTL satisfies the fundamental equivalences laws (shown in Figure 2).

While all logics studied in the next sections satisfy these equivalences, LTL is also satisfying the following variants of the de Morgan-X and, respectively, unwinding laws:

$$\neg X\varphi \equiv_\omega X\neg\varphi$$
$$\varphi R \psi \equiv_\omega \psi \wedge (\varphi \vee X(\varphi R \psi))$$

Moreover, LTL satisfies

$$\mathit{true} \equiv_\omega X\mathit{true}$$

These three equivalences do not hold in each of the logics discussed subsequently—resulting in a partly counterintuitive behaviour of the corresponding logic. To compare equivalences in different logics, we introduce the LTL compliance which states that every equivalence of LTL must hold in the respective logic as well.

DEFINITION 4 (LTL compliance)
We call a linear temporal logic $\mathfrak{L}$ *LTL compliant* iff $\varphi \equiv_\omega \psi$ implies $\varphi \equiv_{\mathfrak{L}} \psi$.

For its importance in various applications, we introduce the *negation normal form* which requires negations only to occur directly in front of atomic propositions.

DEFINITION 5 (Negation Normal Form)
A formula is said to be in *negation normal form*, if $\neg$ only occurs directly in front of atomic propositions, i.e. if the formula is obtained using only the first two rules of Definition 2.

Whenever the de Morgan, de Morgan-X and de Morgan-U/R laws hold, a formula can be translated into negation normal form, and therefore, LTL has a negation normal form.

REMARK 3
Every LTL formula can be transformed into an equivalent formula in negation normal form.

3 LTL on finite traces—existing concepts

Let us now turn our attention to LTLs over finite traces. We start by recalling a finite version of LTL on finite traces as described by Manna and Pnueli [17], here called. We then present Eisner's *et al.*'s [17] *weak* and *strong* versions of LTL on finite traces, denoted with LTL$^-$ and LTL$^+$ respectively, before examining LTL$_3$ [2]. All variants, as we show, provide complementary properties for runtime verification but neither of them satisfies all our maxims put forward in Section 4.

3.1 FLTL

When interpreting LTL formulae over finite traces, the question arises, how to understand $X\varphi$ when a word consists of a single letter, since then, no next position exists on which one is supposed to consider φ. The classical way to deal with this situation, as apparent for example in Kamp's work [15] is to understand X as a *strong* next operator, which is false if no further position exists. Manna and Pnueli suggest in Ref. [17] to enrich the standard framework by adding a dual operator, the weak next $\bar{X}$, which allows to smoothly translate formulae into negation normal form. In other words, the *strong* X operator is used to express with $X\varphi$ that a next state must exist and that this next state has to satisfy property φ. In contrast, the *weak* $\bar{X}$ operator in $\bar{X}\varphi$ says that if there is a next state, then this next state has to satisfy the property φ. We call the resulting logic FLTL defined over the set of LTL formulae (Definition 2) FLTL.

The semantics function $[u \models \varphi]_F$ of FLTL is constructed like the one for standard LTL but with two modifications: if a strong next-state operator in some subformula $X\varphi$ is referring to a state beyond the known finite prefix u, then this subformula $X\varphi$ is evaluated to $\bot$, regardless of φ. Likewise, a subformula $\bar{X}\varphi$, based on the weak next-state operator, always evaluates to $\top$ if it refers to a state beyond u. This approach is extended to the definition of the until and release operators. For example, to satisfy $\varphi U \psi$ with a finite word u, there must exist a position satisfying ψ within u. This concept is explicated in the following definition.

DEFINITION 6 (Semantics of FLTL [17])
Let $u = a_0 \dots a_{n-1} \in \Sigma^*$ denote a finite trace of length n, with $u \neq \epsilon$. The *truth value* of an FLTL formula φ w.r.t. u, denoted with $[u \models \varphi]_F$, is an element of $\mathbb{B}_2$ and is inductively defined as follows: Boolean constants, Boolean combinations and atomic propositions are defined as for LTL (see Figure 1, taking u instead of w). Until/release and (weak) next are defined as shown in Figure 3.

Let us first record that the semantics of FLTL is *not* given for the empty word. As we will see in the discussion of LTL$^{\mp}$, coming up with a semantics w.r.t. the empty word leads to certain particularities that are hence avoided in FLTL. Since $X true \equiv_\omega true$, yet the semantics of FLTL implies that a single letter does satisfy *true* but does not satisfy $X true$, we also find that FLTL is not LTL compliant.

(weak) next

$$[u \models X\varphi]_F = \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

$$[u \models \bar{X}\varphi]_F = \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \top & \text{otherwise} \end{cases}$$

until/release

$$[u \models \varphi\ U\ \psi]_F = \begin{cases} \top & \text{there is a } k \in \{0,\dots n-1\} : [u^k \models \psi]_F = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \leq l < k : [u^l \models \varphi] = \top \\ \bot & \text{else} \end{cases}$$

$$[u \models \varphi\ R\ \psi]_F = \begin{cases} \top & \text{for all } k \in \{0,\dots n-1\} : [u^k \models \psi]_F = \top \text{ or} \\ & \text{there is a } \ \in \{0,\dots n-1\} : [u^k \models \varphi]_F = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \leq l \leq k : [u^l \models \psi] = \top \\ \bot & \text{else} \end{cases}$$

FIGURE 3. Semantics of FLTL formulae over a trace $u = a_0 \dots a_{n-1} \in \Sigma^*$

REMARK 4
The semantics of FLTL formulae is not defined for the empty word.

REMARK 5
FLTL is *not* LTL compliant.

It is easy to see that FLTL adheres to the Boolean laws and satisfies all fundamental equivalence laws: for example, $[u \models \varphi \vee \neg\varphi]_F = [u \models \varphi]_F \sqcup \overline{[u \models \varphi]_F} = \top$. Likewise $[u \models \neg X\varphi]_F = [u \models \bar{X}\neg\varphi]_F$ follows from LTL whenever $|u| > 1$ and from inspecting the semantics in Figure 3 when $|u| = 1$.

REMARK 6
FLTL satisfies the Boolean laws.

REMARK 7
FLTL satisfies all fundamental equivalences laws (Figure 2).

In contrast to LTL, FLTL does not satisfy the following variants of the de Morgan-X and, respectively, unwinding laws, due to the different meaning of X and $\bar{X}$ used in FLTL:

$$\begin{aligned} \neg X\varphi &\not\equiv_F X\neg\varphi \\ \varphi R \psi &\not\equiv_F \psi \wedge (\varphi \vee X(\varphi R \psi)) \end{aligned}$$

But since de Morgan laws are satisfied by FLTL, a negation normal form still exists for all properties.

REMARK 8
Every FLTL formula can be transformed into an equivalent formula in negation normal form.

3.2 LTL$^+$ and LTL$^-$

In Ref. [8], Eisner *et al*. propose two entwined versions of LTL on finite traces, reflecting a *weak* and a *strong* view. We call the resulting logics LTL$^-$and LTL$^+$, respectively. If we speak of both logics at the same time, we also write LTL$^\mp$.

Recall that FLTL is not defined for the empty trace. Provided that the semantics for formulae w.r.t. the empty word is well-defined, one no longer needs to consider the semantics of next and until operators differently than in LTL, as done in FLTL: having a semantics for the empty word, the evaluation of $X\varphi$ for a word of length 1 can use the semantic value of φ w.r.t. the empty word. Likewise, for $\varphi\, U\, \psi$ and a word u, one no longer has to require that $i<|u|$ when considering whether u^i satisfies ψ—in case $i \geq |u|$, one considers ψ w.r.t. the empty word.

LTL$^\mp$ provides a semantics for the empty word, by actually taking two approaches: in the weak view (LTL$^-$), every formula is satisfied by the empty word. Dually, in the strong view (LTL$^+$), the empty word does not satisfy any formula. This approach leads to certain particularities: for example, in the weak view, the empty word satisfies *false*, while in the strong view, it does not satisfy *true*. Moreover, the strong and weak view have be *entwined* to establish a sound meaning in the context of negation: $[\epsilon \models \neg\varphi]_+$ should be $\bot$ (taking the strong view) while at the same time $\overline{[\epsilon \models \varphi]_+} = \overline{\bot}$ should hold, implying that $\bot = \top$. LTL$^\mp$ overcomes this difficulty, by toggling between the weak and strong view whenever negation occurs, i.e. $[u \models \neg\varphi]_+ = \overline{[u \models \varphi]_-} = \overline{\top}$ (also for $u=\epsilon$).

Let us now introduce LTL$^\mp$ formally. As syntax, we consider again formulae of *LTL*. Though [8] does not provide *true* and *false* and no dual operators, these can be added with the appropriate semantics easily.

DEFINITION 7 (Semantics of LTL$^\mp$ [8])
Let $u = a_0 \ldots a_{n-1} \in \Sigma^*$ denote a finite trace of length n. The *truth value* of an LTL$^\mp$ formula φ w.r.t. u, denoted with $[u \models \varphi]_\mp$, is an element of $\mathbb{B}_2$ and is inductively defined as follows: Boolean constants, Boolean combinations, and atomic propositions are defined as shown in Figure 4, while the semantics for the remaining formulae is as in LTL (Figure 1, taking u instead of w).

To compare LTL$^\mp$ with FLTL, and later with LTL$_3$, we check which of the discussed properties are satisfied by LTL$_3$ as well.

REMARK 9
The semantics of LTL$^\mp$ formulae is defined for the empty trace.

We can easily check that LTL$^\mp$ indeed matches our intuition:

$$\begin{aligned}[u \models \varphi]_- &= \top, \text{ if } u=\epsilon\\ [u \models \varphi]_+ &= \bot, \text{ if } u=\epsilon\end{aligned}$$

We write $\varphi \equiv_\mp \psi$ to denote that φ is both weak as well as strong equivalent to ψ, i.e. that $\varphi \equiv_+ \psi$ and $\varphi \equiv_- \psi$ hold. In the weak view, *false* holds for the empty trace, and, in the strong view, *true* does not hold for the empty trace. Interestingly, however, we have

$$\begin{aligned}\varphi \vee \neg\varphi &\equiv_\mp \mathit{true}\\ \varphi \wedge \neg\varphi &\equiv_\mp \mathit{false}\end{aligned}$$

In consequence, for φ being p, we see that the Boolean laws are *not* satisfied by LTL$^\mp$, as the semantics of *true* is not $\top$ for the empty word.

Boolean constants

$$[u \models true]_- = \top \qquad [u \models false]_- = \begin{cases} \top & \text{if } u = \epsilon \\ \bot & \text{else} \end{cases}$$

$$[u \models true]_+ = \begin{cases} \top & \text{if } u \neq \epsilon \\ \bot & \text{else} \end{cases} \qquad [u \models false]_+ = \bot$$

Boolean combinations

$$\begin{aligned} [u \models \neg\varphi]_- &= \overline{[u \models \varphi]_+} \\ [u \models \neg\varphi]_+ &= \overline{[u \models \varphi]_-} \\ [u \models \varphi \vee \psi]_\mp &= [u \models \varphi]_\mp \sqcup [u \models \psi]_\mp \\ [u \models \varphi \wedge \psi]_\mp &= [u \models \varphi]_\mp \sqcap [u \models \psi]_\mp \end{aligned}$$

atomic propositions

$$[u \models p]_- = \begin{cases} \top & \text{if } u = \epsilon \text{ or } p \in a_0 \\ \bot & \text{else} \end{cases} \qquad [u \models p]_+ = \begin{cases} \top & \text{if } u \neq \epsilon \text{ and } p \in a_0 \\ \bot & \text{else} \end{cases}$$

$$[u \models \neg p]_- = \begin{cases} \top & \text{if } u = \epsilon \text{ or } p \notin a_0 \\ \bot & \text{else} \end{cases} \qquad [u \models \neg p]_+ = \begin{cases} \top & \text{if } u \neq \epsilon \text{ and } p \notin a_0 \\ \bot & \text{else} \end{cases}$$

FIGURE 4. Semantics of LTL$^\mp$ formulae over a trace $u = a_0 \ldots a_{n-1} \in \Sigma^*$

REMARK 10
LTL$^\mp$ does not satisfy the Boolean laws.

Moreover, this implies that *true* is not equivalent in general to X*true*: while a single letter does satisfy *true* in the strong view, it does *not* satisfy X*true*. This implies:

REMARK 11
LTL$^\mp$ is *not* LTL compliant.

It is easy to see that LTL$^\mp$ satisfies all fundamental equivalence laws.

REMARK 12
LTL$^\mp$ satisfies all fundamental equivalences laws (Figure 2).

As de Morgan laws are satisfied by LTL$^\mp$, the existence of the negation normal form follows:

REMARK 13
Every LTL$^\mp$ formula can be transformed into an equivalent formula in negation normal form.

3.3 LTL$_3$

In Refs [1, 2], we proposed LTL$_3$ as an LTL logic with a semantics for finite traces, which follows the idea that a finite trace is a prefix of a so-far unknown infinite trace. More specifically, LTL$_3$ uses the standard syntax of LTL as defined in Definition (2) but employs a semantics function $[u \models \varphi]_3$ which evaluates each formula φ and each finite trace u of length n to one of the truth values in $\mathbb{B}_3 = \{\top, \bot, ?\}$. $\mathbb{B}_3 = \{\top, \bot, ?\}$ is defined as a de Morgan lattice with $\bot \sqsubset ? \sqsubset \top$, and with $\bot$ and $\top$ being complementary to each other while ? being complementary to itself.

The idea of the semantics for LTL$_3$ is as follows: if every infinite trace with prefix u evaluates to the same truth value $\top$ or $\bot$, then $[u \models \varphi]_3$ also evaluates to this truth value. Otherwise $[u \models \varphi]_3$

evaluates to ?, i.e. we have $[u \models \varphi]_3 = ?$ if different continuations of u yield different truth values. This leads to the following definition:

DEFINITION 8 (Semantics of LTL$_3$)
Let $u = a_0 \ldots a_{n-1} \in \Sigma^*$ denote a finite trace of length n. The *truth value* of a LTL$_3$ formula φ w.r.t. u, denoted with $[u \models \varphi]_3$, is an element of $\mathbb{B}_3$ and defined as follows:

$$[u \models \varphi]_3 = \begin{cases} \top & \text{if } \forall w \in \Sigma^\omega : [uw \models \varphi]_\omega = \top \\ \bot & \text{if } \forall w \in \Sigma^\omega : [uw \models \varphi]_\omega = \bot \\ ? & \text{otherwise.} \end{cases}$$

As opposed to the logics introduced so far, LTL$_3$'s semantics is *not* defined in an inductive manner, i.e. the semantics of a formula is *not* given by the meaning of its subformulae—for good reasons: consider a proposition p with respect to the empty word ϵ. In LTL$_3$, we have $[\epsilon \models p]_3 = ? = [\epsilon \models \neg p]_3$. The join of these values is thus ?. But in contrast, $[\epsilon \models p \vee \neg p]_3 = \top$ holds, since $p \vee \neg p$ is a tautology.[1] In other words, we have:

REMARK 14
The semantics of LTL$_3$ cannot be defined inductively on the structure of the formula.

Let us first recall that LTL$_3$ is a also defined for the empty word.

REMARK 15
The semantics of LTL$_3$ formulae is defined for the empty word.

As LTL$_3$'s semantics is derived from LTL's semantics, we get that it is LTL compliant, as opposed to FLTL and LTL$^{\mp}$.

REMARK 16
LTL$_3$ *is* LTL compliant.

As moreover *true* is mapped to $\top$, we get

REMARK 17
LTL$_3$ satisfies the Boolean laws.

Finally, LTL$_3$ satisfies all fundamental equivalence laws and henceforth, a negation normal form exists for all properties.

REMARK 18
LTL$_3$ satisfies all fundamental equivalence laws (Figure 2).

REMARK 19
Every LTL$_3$ formula can be transformed into an equivalent formula in negation normal form.

4 Maxims for runtime verification of LTL derivates

Since our original motivation is to validate common LTL-specified properties by means of runtime verification, we want to compare the applicability of logics in this context. To do so, we need to determine a frame of reference for those logics to be considered. Hence, we assume that each logic

[1]Note, we call a formula φ a tautology, if $[w \models \varphi] = \top$ for every w.

in concern is able to express *syntactically* all LTL properties, and moreover, we assume that the semantics evaluation function $[u \models \varphi]$ of the logic in concern maps each finite and non-empty word $u = a_0 a_1 \ldots a_{n-1} \in \Sigma^*$ together with a formula φ to a value from a truth domain (Definition 1), such that the following rules hold for each non-empty finite word u[2]

$$\begin{array}{lll}
[u \models true] & = \top & \\
[u \models p] & = \top & \iff p \in a_0 \\
[u \models \varphi_1 \vee \varphi_2] & = [u \models \varphi_1] \sqcup [u \models \varphi_2] & \\
[u \models X\varphi] & = \top & \Longleftarrow \quad |u| > 1 \text{ and } [u^1 \models \varphi] = \top \\
[u \models \varphi\, U\, \psi] & = [u \models \psi] \sqcup ([u \models \varphi] \sqcap [u \models X U \psi]) &
\end{array}$$

This minimal set of requirements on $[u \models \varphi]$ is established as the set of rules commonly shared by the logics discussed in the previous section, i.e. FLTL [17], LTL$^{\mp}$ [8], as well as LTL$_3$ [2]. The above rules deviate from standard LTL in the following two respects:

(a) If $|u| > 1$ and $[u^1 \models \varphi] = \top$ holds, then we require $[u \models X\varphi] = \top$ to hold—however, in contrast to LTL, we do not require the converse.

(b) Negation is not guaranteed to yield a complementary truth value, i.e. we do not require $[u \models \varphi] = \overline{[u \models \neg\varphi]}$.

Starting from these core rules, we introduce four maxims which we consider essential for each semantics definition for LTL on finite traces *aimed at runtime verification.* The first two of them, Maxims (1) and (2), mimic and reintroduce the LTL evaluation rules which we dropped in (a) and (b)—as far as possible since we consider a semantics for LTL (on finite traces) without these rules counterintuitive:

(1) *existential next* requires the inclusion of a strong next operator, and

(2) *complementation by negation* requires that a negated formula evaluates to the complemented and different truth value.

Since it is impossible to turn condition in (a) into an 'if and only if', we formulate Maxim (1) to require the next operator to behave as in LTL in as many cases as possible: since we consider finite traces, we have to handle the semantics of X when it refers to a state beyond the currently available finite trace. To reconstitute (b), Maxim (2) reintroduces and generalizes the corresponding rule of the standard semantics on infinite traces to multi-valued Boolean domains by applying the complement operator of the underlying Boolean domain.

The remaining two Maxims (3) and (4) relate the semantics on finite traces with the standard LTL semantics by considering all possible continuations of the evaluated finite traces:

(3) *impartiality* requires that a finite trace is not evaluated to $\top$ ($\bot$) if there still exists an infinite continuation leading to another verdict, and

(4) *anticipation* requires that once every infinite continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this very same verdict.

By means of these four maxims, we evaluate the logics discussed in the preceding sections, i.e. FLTL [17] the logics LTL$^{\mp}$ [8], and LTL$_3$ [2]. As all these logics fail to satisfy all four maxims simultaneously, we introduce in Section 5 a *four-valued* logic resolving the requirements imposed by the four maxims.

[2]The restriction to *non-empty* finite words is necessary since the semantics of FLTL is not defined on empty words and because $[\epsilon \models true]_+ \neq \top$ in LTL$^+$.

4.1 Approximating LTL semantics

We start with the first two maxims which re-establish the semantics of the next-state operator and the meaning of negation. Afterwards, we discuss the combination of these two maxims, since their combination raises some further issues.

4.1.1 Existential next

As discussed in Ref. [17], the difficulty for an LTL semantics on finite traces lies in the next-state operator X. Given a string $u = a_0$ consisting of a single symbol, the question is which semantics to choose for $X\varphi$ as the formula refers to an unavailable state:

$$[u \models X\varphi] = ? \text{ for } |u| = 1 \tag{1}$$

We follow the approach of Ref. [17] in understanding the next-state operator as an operator that first assures that there exists a next state second satisfies φ. We only allow one exception to this rule: if φ is a tautology, then $[u \models X\varphi]$ is also allowed to evaluate to $\top$. Without this exception, any logic satisfying our demands, would have to satisfy $[u \models X true] \neq \top$ for every word u consisting of a single symbol, which we consider counter intuitive in the setting of runtime verification. This consideration leads to the first of our four maxims:

Maxim 1 (Existential Next)
A logic adheres to the *existential next* maxim, if for every property φ and every finite word $u \in \Sigma^*$ with $[u \models X\varphi] = \top$, one of the following two conditions holds: either φ is an LTL-tautology or there exists a next state, i.e. $|u| > 1$, such that this next state satisfies φ, i.e. we require

$$\begin{aligned}[u \models X\varphi] = \top \implies & |u| > 1 \text{ and } [u^1 \models \varphi] = \top \text{ or} \\ & |u| = 1 \text{ and } \forall w \in \Sigma^\omega [w \models \varphi]_\omega = \top\end{aligned}$$

Now, considering FLTL, we find that its semantics $[u \models X\varphi]_F$ is even more restrictive: $[u \models X\varphi]_F = \top$ implies that a next state u^1 exists and this next state u^1 satisfies φ, i.e.

$$[u \models X\varphi]_F = \top \implies |u| > 1 \text{ and } [u^1 \models \varphi]_F = \top$$

holds. Next, the *strong* view of Ref. [8] defines a *strong* next operator—in precisely the same way as FLTL. On the other hand, in the weak variant LTL$^-$of [8], the next operator behaves dually, i.e. $[u \models X\varphi]_- = \top$ iff either $u = \epsilon$ or $[u^1 \models \varphi]_- = \top$, i.e.

$$[u \models X\varphi]_- = \top \implies |u| = 0 \text{ or } [u^1 \models \varphi]_- = \top$$

Hence, LTL$^+$does satisfy Maxim (1) but LTL$^-$does not satisfy Maxim (1). Finally, the semantics of LTL$_3$ is matching Maxim (1) since whenever a formula $X\varphi$ evaluates to $\top$ in LTL$_3$, it must evaluate to $\top$ in LTL for every possible infinite continuation. Thus, we have $[u \models X\varphi]_3 = \top$ iff for all $w \in \Sigma^\infty$ $[uw \models X\varphi]_\omega = \top$, which is equivalent to, for all $w \in \Sigma^\infty$ $[u^1 w \models X\varphi]_\omega = \top$. If $|u| = 1$, this means that all traces satisfy φ, while for $|u| > 1$, we get $[u \models X\varphi]_3 = [u^1 \models \varphi]_3$. Thus,

$$\begin{aligned}[u \models X\varphi] = \top \implies & |u| > 1 \text{ and } [u^1 \models \varphi] = \top \text{ or} \\ & |u| = 1 \text{ and } \forall w \in \Sigma^\omega [w \models \varphi]_\omega = \top\end{aligned}$$

4.1.2 Complementation by negation

Our second maxim states that a negated formula indeed yields the complemented truth value of the original formula which must differ from the original one. We consider any other choice to be grossly counterintuitive since many fundamental relationships rely on complementary truth values for formulae and their negations. Furthermore, the requirement that the complemented truth value is different from the original one ensures that the logic does not blur the semantical evaluation into a single and non-discriminative truth value.

MAXIM 2 (Complementation by Negation)
A logic adheres to the *complementation by negation* maxim, if a property φ and its complement $\neg\varphi$ yield complementary and different truth values for all finite words $u \in \Sigma^*$, i.e. for each property φ and each finite word u,

$$[u \models \varphi] = \overline{[u \models \neg\varphi]} \text{ and } [u \models \varphi] \neq [u \models \neg\varphi]$$

must hold.

Recalling the logics from Section 3, we find that FLTL adheres in its definition to the requirements of Maxim (2), since we have

$$[u \models \neg\varphi]_F = \overline{[u \models \varphi]_F}$$

and since FLTL is defined over a two-valued truth domain where complementary values always differ. In case of LTL$^+$, Maxim (2) does not hold, as can be seen in the following example with $|u| = 1$,

$$[u \models \neg Xp]_+ = \overline{[u \models Xp]_-} = \overline{\top} = \bot = [u \models Xp]_+$$

which holds dually for LTL$^-$, as well. LTL$_3$ is also not satisfying Maxim (2) since it collapses the truth values for the unforeseen future into its inconclusive truth value. It holds that

$$[u \models \varphi]_3 = ? = [u \models \neg\varphi]_3 \text{ for all } \varphi \text{ and } u \text{ with } [u \models \varphi]_3 = ?$$

since whenever $[u \models \varphi]_3 = ?$ holds, there must exist two infinite continuations $w \neq w' \in \Sigma^\omega$ such that $[uw \models \varphi]_\omega = \top$ and $[uw' \models \varphi]_\omega = \bot$ hold, leading to $[uw \models \neg\varphi]_\omega = \bot$ and $[uw' \models \varphi]_\omega = \top$ such that $[u \models \neg\varphi]_3 = ?$ holds as well.

But still, for $[u \models \varphi] = \top$ with all infinite continuations $w \in \Sigma^\omega$ yielding $[uw \models \varphi]_\omega = \top$, we find $[uw \models \neg\varphi]_\omega = \bot$ and hence $[u \models \neg\varphi] = \bot$, such that we have

$$[u \models \varphi]_3 = \overline{[u \models \neg\varphi]_3} \tag{2}$$

Combining Maxims (1) and (2): Remember that we introduced Maxims (1) and (2) in order to re-establish the original LTL semantics on infinite words as much as possible. To enable an engineer to deal with LTL-specified properties intuitively, some well-known and important equivalences should hold in logics aimed at runtime verification, too. Not surprisingly, the combination of Maxim (1), demanding an existential next, and Maxim (2), requiring negation to correspond to complementation, leads to difficulties in

$$\neg X\varphi \not\equiv X\neg\varphi$$

To see this, assume that φ is a formula which is neither unsatisfiable nor a tautology with respect to LTL. Then by Maxim (1), we have $[u \models X\varphi] \neq \top$ for every $|u| = 1$ and hence by Maxim (2)

$[u \models \neg X\varphi] \neq \bot$ must hold. At the same time, Maxim (1) requires $[u \models X\neg\varphi] \neq \top$ and hence we are left with the following options:

(a) To use a two-valued semantics and accept $\neg X\varphi \not\equiv X\neg\varphi$ to hold, breaking with the intuitive understanding of LTL properties.
(b) To stick to the equivalence $\neg X\varphi \equiv X\neg\varphi$, but to use a multi-valued semantics with $[u \models \neg X\varphi] = [u \models X\neg\varphi] \notin \{\top, \bot\}$ (for $|u| = 1$)—hereby violating Maxim (2).
(c) To distinguish between a *strong* (denoted with X) and a *weak* version (denoted with $\bar{X}$) of the next-state operator with

$$[u \models \bar{X}\varphi] = \top \quad \iff \quad |u| = 1 \text{ or } [u^1 \models \varphi] = \top$$

and leading to

$$\neg X\varphi \equiv \bar{X}\neg\varphi$$

which remedies the original equivalence to a certain extent, while adhearing to Maxims (1) and (2). We call the strong next-state operator X also *existential* next-state operator, as it requires a next-state to exist, and the weak next-state operator $\bar{X}$ *universal* next-state operator.

Summarizing the discussion above, we remark that any logic adhearing to both Maxims (1) and (2) cannot be LTL compliant. In other words, depeding on the application, one has to choose between the Maxims (1) and (2) on the one hand side and LTL compliance on the other side.

The introduction of a strong and a weak version of the next-state operator additionally allows to cope with the intuitive meaning of LTL's finally and globally operators:

Intuitively, the finally operator F is of existential nature [14], as some property should eventually be shown, while the globally operator G is of universal character as something should hold in every position of a word. Accordingly, $F\varphi$ should evaluate to false if φ does not hold in the current state and nothing is known about the future, while $G\varphi$ should become true, if φ holds in the current state and nothing is known about the successor states.

Note that in LTL, we have $F\varphi \equiv \varphi \vee XF\varphi$ as well as $G\varphi \equiv \varphi \wedge XG\varphi$. Consequently, $XF\varphi$ should be false, if no subsequent state exists, while $XG\varphi$ should be true in the same situation. This contradiction can be resolved with the addition of the universal next-state operator $\bar{X}$. Following option (c), we can rewrite the above LTL equivalences as $F\varphi \equiv \varphi \vee XF\varphi$ and as $G\varphi \equiv \varphi \wedge \bar{X}G\varphi$.

4.2 *Dealing with the future*

The so far developed view is meaningful in a setting which is only concerned with *completed* or *terminated* paths. In runtime verification, however, we are given a *finite prefix* of a continuously expanding trace. Therefore, it is clear that there will be a next state—this continuation is just not known yet. To reflect this situation, we postulate two further maxims for logics suitable for runtime verification.

4.2.1 Impartiality

The first maxim says that the semantics never evaluates to true or false prematurely. In the *impartiality* maxim, we require a semantics never evaluates a property φ and a finite word $u \in \Sigma^*$ to $\top$ ($\bot$), as long as there exists a continuation $w \in \Sigma^\omega$ such that $[uw \models \varphi]_\omega \neq \top$ ($[uw \models \varphi]_\omega \neq \bot$) holds.

MAXIM 3 (Impartiality)
A logic adheres the *impartiality* maxim, if for each property φ and each finite word $u \in \Sigma^*$,

$$\begin{array}{lll} [u \models \varphi] = \top & \Longrightarrow \forall w \in \Sigma^\omega & [uw \models \varphi]_\omega = \top \\ [u \models \varphi] = \bot & \Longrightarrow \forall w \in \Sigma^\omega & [uw \models \varphi]_\omega = \bot \end{array}$$

is satisfied.

Note that no two-valued logic can possibly satisfy Maxim (3) since each such logic must evaluate every property φ with uncertain outcome to either $\top$ or $\bot$. For example, in every logic satisfying Maxim (3) we must have $[u \models Xp] \notin \{\top, \bot\}$ for $|u| = 1$.

Turning again to the logics from Section 3, we find that FLTL and LTL$^{\mp}$ are all two-valued logics, and henceforth, they do not satisfy Maxim (3). The definition of the semantics of LTL$_3$ on the other hand directly matches the requirements of Maxim (3), as it has been defined with this maxim in mind.

4.2.2 Anticipation

When considering $Xtrue$ on a string $u = a_0$ of length 1, there is no reason to evaluate $Xtrue$ to any truth value other than $\top$ since every possible continuation will satisfy $Xtrue$. We therefore postulate that the semantics should be as anticipatory as possible. We say that a logic satisfies the *anticipation* maxim, if its semantics always evaluates a property φ and a finite word $u \in \Sigma^*$ to $\top$ ($\bot$), once there exists no continuation $w \in \Sigma^\omega$ such that $[uw \models \varphi]_\omega \neq \top$ ($[uw \models \varphi]_\omega \neq \bot$) holds.

MAXIM 4 (Anticipation)
A logic adheres the *anticipation* maxim, if for each property φ and each finite word $u \in \Sigma^*$, the following holds:

$$\begin{array}{lll} [u \models \varphi] = \top & \Longleftarrow \forall w \in \Sigma^\omega & [uw \models \varphi]_\omega = \top \\ [u \models \varphi] = \bot & \Longleftarrow \forall w \in \Sigma^\omega & [uw \models \varphi]_\omega = \bot \end{array}$$

Maxim (4) is not satisfied by FLTL, since $[u \models Xtrue]_F = \bot$ holds for $|u| = 1$. Similarly LTL$^{\mp}$ does not satisfy Maxim (4), as demonstrated by the two dual examples $[u \models Xtrue]_+ = \bot$ and $[u \models Xfalse]_- = \top$, again for $|u| = 1$. Finally, since Maxim (4) [as well as Maxim (3)] was instrumental in the definition of LTL$_3$, its semantics directly reflects and hence satisfies Maxim (4).

5 RV-LTL

In the previous section, we established four maxims to be satisfied by an LTL-derived logic which is aimed at runtime verification applications. Moreover, we analysed the logics introduced in Section 3 in terms of these maxims and found that none of these logics adheres to all four maxims.

To overcome this situation, we develop in this section a new logic, called *RV-LTL*. As in case of all other logics discussed in this article, we use the set of LTL formulae (Definition 2) to define the syntax of RV-LTL. Since all logics in this article are formulated atop this very same set of formulae, it is possible to combine the semantical concepts of these logics as well. Observing that LTL$_3$ satisfies Maxims (1), (3) and (4), whereas FLTL satisfies Maxim (2), we design the semantics of RV-LTL as a combination of the semantics of LTL$_3$ and FLTL.

LTL$_3$ *matches Maxim (1)* since it only evaluates a formula and a finite prefix to $\top$ if this formula will be satisfied by any possible continuation of the prefix, and that LTL$_3$ *satisfies Maxim (3) and (4)* by construction—since LTL$_3$ has been defined with these two maxims in mind.

On the other hand, LTL$_3$ does *not satisfy Maxim (2)*, since it blurs every uncertain situation into a single inconclusive verdict: a finite word $u \in \Sigma^*$ and a formula φ is evaluated to $[u \models \varphi]_3 = ?$ whenever $[uw \models \varphi]_\omega \neq [uw' \models \varphi]_\omega$ holds for two infinite continuations $w \neq w' \in \Sigma^\omega$. Since this case is arising frequently in practically important properties, the choice made in the definition of LTL$_3$ is unsatisfactory. Consider for example the standard *request/acknowledge property*

$$\varphi \equiv G(r \to Fa)$$

which states that all requests must be acknowledged eventually: for every finite prefix u, we have that $[ur^\omega \models \varphi]_\omega = \bot$ and $[ua^\omega \models \varphi]_\omega = \top$ where r^ω and a^ω are infinite continuations repeating r and a ad infinitum. Therefore $[u \models \varphi]_3 = [u \models \neg\varphi]_3 = ?$ holds for *all* finite words u.

Looking for a more expressive and discriminative semantics, we followed the intuition that for the property $\varphi \equiv G(r \to Fa)$, we would like to have a semantics such that

– a finite string ending in a (i.e. all requests have been acknowledged) yields a truth value which indicates that φ is probably satisfied; whereas,
– a finite string ending in r (i.e. there is a request not acknowledged yet) should evaluate to a truth value which expresses that φ is likely to remain unsatisfied.

The reason for these choices are as follows: given a finite string ending in a, *all* past requests have been served and therefore it remains to check that *all* future occurrences will be served as well. Since no request is pending and the universal globally operator is dominating, we would expect the trace to be interpreted as a 'presumably true' one. In case of a string ending in r, we know that there must *exist* a future occurrence of a in order to satisfy φ. Since we have a request pending and the existential eventually operator is dominating, we would expect the trace to be interpreted as a 'presumably false' one.

This intuition is readily expressed by the semantics of FLTL which is based upon a strong and a weak next operator. As will be discussed subsequently in this section, the existential nature of the strong next operator X translates into an existential semantics for the eventually operator F while the universal character of the weak next operator $\bar{X}$ leads to a universal semantics for the globally operator G.

The problem with FLTL is that it does not distinguish between prefixes which lead to *presumably* true or false continuations and prefixes which lead to *certainly* true of false continuations, i.e. FLTL does not satisfy Maxim (3) and (4). Henceforth, we define RV-LTL as a logic which *combines* the semantics of LTL$_3$ and FLTL.

5.1 *Semantics of RV-LTL*

To express the truth values true, false, presumably true and presumably false, we use a four-valued semantics for RV-LTL with $\mathbb{B}_4 = \{\bot, \bot^p, \top^p, \top\}$ as the set of truth values. Ordering $\bot \leq \bot^p \leq \top^p \leq \top$, the operators $\sqcap$ and $\sqcup$ are then defined as expected. To obtain a de Morgan lattice and thus a truth domain, $\bot$ and $\top$ are defined to be complementary to each other as well as $\bot^p$ and $\top^p$, where complementation is denoted with $\bar{\ }$. Note that $\mathbb{B}_4$ is not a Boolean lattice, as, for example, $\bot^p \sqcup \overline{\bot^p} = \bot^p \sqcup \top^p \neq \top$. Thus, it is different from the (unique) Boolean lattice with four elements, which is often considered in the context of multi-valued logics [5]. However, the distributive laws hold:

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$

$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$$

According to the above discussion, we take the truth value of LTL$_3$, whenever it is conclusive, i.e. whenever it is either $\top$ or $\bot$. If LTL$_3$ provides an inconclusive verdict (?) only, we resort to FLTL to settle a more discriminative choice. In this case, if FLTL leads to a $\top$ ($\bot$) verdict, RV-LTL evaluates to $\top^p$ ($\bot^p$).

Note that, since the semantics of FLTL is undefined on the empty word (Remark 4), the semantics of RV-LTL remains undefined on the empty word as well (except for properties which are LTL equivalent to either *true* or *false*).

Definition 9 (Semantics of RV-LTL)
Let $u = a_0 \dots a_{n-1} \in \Sigma^*$ denote a finite and non-empty trace of length $n = |u|$. The *truth value* of an RV-LTL formula φ w.r.t. u, denoted with $[u \models \varphi]_{RV}$, is an element of $\mathbb{B}_4$ and is defined as follows:

$$[u \models \varphi]_{RV} = \begin{cases} \top & \text{if } [u \models \varphi]_3 = \top \\ \bot & \text{if } [u \models \varphi]_3 = \bot \\ \top^p & \text{if } [u \models \varphi]_3 = ? \text{ and } [u \models \varphi]_F = \top \\ \bot^p & \text{if } [u \models \varphi]_3 = ? \text{ and } [u \models \varphi]_F = \bot \end{cases}$$

The semantics of RV-LTL as given in Definition 9 directly provides an efficient way to construct a monitor procedure for RV-LTL: by running a monitor for LTL$_3$ and one for FLTL simultaneously and by combining their respective results following Definition 9, we obtain a monitor procedure for RV-LTL. We exploit this fact in the Section 6, where we discuss the monitor construction for RV-LTL in detail.

Before discussing the viability of Definition 9, let us remark explicitly that RV-LTL's semantics is a refinement of LTL$_3$'s semantics. Consequently, the semantics of LTL$_3$ is expressible by mapping a $\top^p/\bot^p$ value to ?:

Remark 20
Let $u \in \Sigma^*$ be a finite and non-empty trace and let φ be an LTL$_3$ formula. Then the following holds

$$[u \models \varphi]_3 = \begin{cases} \top & \text{if } [u \models \varphi]_{RV} = \top \\ \bot & \text{if } [u \models \varphi]_{RV} = \bot \\ ? & \text{if } [u \models \varphi]_{RV} \in \{\top^p, \bot^p\} \end{cases}$$

Let us now consider the properties of RV-LTL in the same manner as done for the other temporal logics. First, since the semantics of FLTL is undefined on the empty word, the semantics of RV-LTL is undefined on the empty word as well.

Remark 21
The semantics of RV-LTL formulae is undefined for the empty word.

Next, note that it is impossible is to define the semantics of RV-LTL inductively, since this is already impossible for LTL$_3$ (Remark 14).

Remark 22
The semantics of RV-LTL cannot be defined inductively on the structure of the formula.

Likewise, we get from LTL$_3$ that RV-LTL satisfies the Boolean laws.

REMARK 23
RV-LTL satisfies the Boolean laws.

Combining the results from LTL$_3$ and FLTL, we get

REMARK 24
RV-LTL satisfies all fundamental equivalences laws (Figure 2). Thus, every RV-LTL formula can be transformed into an equivalent formula in negation normal form.

Using that $\neg X\varphi \not\equiv_F X\neg\varphi$ in FLTL, we learn that RV-LTL, as opposed to LTL$_3$, is *not* LTL compliant.

REMARK 25
RV-LTL is *not* LTL compliant.

5.2 RV-LTL implements our Maxims

RV-LTL's semantics satisfies all four maxims: RV-LTL adheres Maxims (1), (3) and (4) since LTL$_3$ does so. in case of Maxim (1), we need to consider the case $[u \models X\varphi]_{RV} = \top$, which can only happen if $[u \models \varphi]_3 = \top$ which implies in turn that every continuation $w \in \Sigma^\omega$ leads to a positive verdict, i.e. $[uw \models \varphi]_\omega = \top$—matching the requirements of Maxim (1).

To see that Maxims (3) and (4) are satisfied by RV-LTL, observe that

$$[u \models \varphi]_{RV} \in \{\top, \bot\} \text{ iff } [u \models \varphi]_{RV} = [u \models \varphi]_3 \text{ iff } [u \models \varphi]_3 \in \{\top, \bot\}$$

In case of Maxim (3), we can ignore all cases with $[u \models \varphi]_{RV} \neq \{\top, \bot\}$, such that the semantics of RV-LTL and LTL$_3$ coincides in all remaining cases—and since LTL$_3$ satisfies Maxim (3), RV-LTL does as well. Finally, in case of Maxim (4), we can ignore all cases with $[u \models \varphi]_3 \notin \{\top, \bot\}$, and again, in this case the semantics of RV-LTL and LTL$_3$ coincides—and hence RV-LTL adheres to Maxim (4) since LTL$_3$ does so.

It remains to show that RV-LTL satisfies Maxim (2). First note that complementary truth values in $\mathbb{B}_4$ are always different, i.e. $t \neq \bar{t}$ for all $t \in \mathbb{B}_4$. Thus, we only need to prove that $[u \models \varphi]_{RV} = \overline{[u \models \neg\varphi]_{RV}}$ holds for all finite words u and all properties φ. Assume that $[u \models \varphi]_{RV} = \top$ holds. Then we have $[u \models \varphi]_3 = \top$ and therefore by Equation (2) $[u \models \neg\varphi]_3 = \bot$ and hence by Definition 9 $[u \models \neg\varphi]_{RV} = \bot$. Dually, we find that $[u \models \varphi]_{RV} = \bot$ implies that $[u \models \neg\varphi]_{RV} = \top$. Now assume that $[u \models \varphi]_{RV} \in \{\top^p, \bot^p\}$ holds. In this case we have $[u \models \varphi]_3 = ?$ and therefore by Equation (2) $[u \models \neg\varphi]_3 = ?$ as well. But then the RV-LTL semantics of u with respect to both, φ and $\neg\varphi$ are determined by the FLTL semantics, i.e. $[u \models \varphi]_{RV} = [u \models \varphi]_F = \overline{[u \models \neg\varphi]_F} = \overline{[u \models \neg\varphi]_{RV}}$.

5.3 RV-LTL and request/acknowledge properties

Let us reconsider the motivating example for RV-LTL:

$$G(r \to Fa)$$

Recall that the finally operator F and globally operator G are defined as abbreviations $F\varphi := true\, U\, \varphi$ and $G\varphi := \neg F \neg\varphi$. Using the equivalences from Figure 2 we get that

$$\begin{aligned} F\varphi &\equiv_{RV} true\, U\, \varphi && \textit{(definition)} \\ &\equiv_{RV} \varphi \vee (true \wedge X(true\, U\, \varphi)) && \textit{(unwinding)} \\ &\equiv_{RV} \varphi \vee XF\varphi && \textit{(tertium-non-datur)} \end{aligned}$$

and

$$\begin{aligned}
G\varphi &\equiv_{RV} \neg F\neg\varphi && \textit{(definition)}\\
&\equiv_{RV} \neg(\mathit{true}\,U\,\neg\varphi) && \textit{(definition)}\\
&\equiv_{RV} \neg(\neg\varphi\vee(\mathit{true}\wedge X(\mathit{true}\,U\,\neg\varphi)) && \textit{(unwinding)}\\
&\equiv_{RV} \neg\neg\varphi\wedge\neg(\mathit{true}\wedge X(\mathit{true}\,U\,\neg\varphi)) && \textit{(de Morgan)}\\
&\equiv_{RV} \varphi\wedge\neg(X(\mathit{true}\,U\,\neg\varphi)) && \textit{(de Morgan)}\\
&\equiv_{RV} \varphi\wedge\bar{X}\neg(\mathit{true}\,U\,\neg\varphi)) && \textit{(de Morgan-X)}\\
&\equiv_{RV} \varphi\wedge\bar{X}G\varphi && \textit{(definition)}
\end{aligned}$$

Thus, we obtain the two equivalences $F\varphi\equiv_{RV}\varphi\vee XF\varphi$ and $G\varphi\equiv_{RV}\varphi\wedge\bar{X}G\varphi$. $F\varphi\equiv_{RV}\varphi\vee XF\varphi$ reflects that φ must be satisfied in the future: if φ is not satisfied immediately, then there must be a satisfying future state. If no such future state exists, the formula evaluates to $\bot^p$—unless the formula evaluates to one of $\{\top,\bot\}$, in which case the future is not important. Similarly, $G\varphi\equiv\varphi\wedge\bar{X}G\varphi$ shows that φ must be satisfied in the current state and in all observable future states. If we do not know the future, the formula evaluates to $\top^p$—again, unless the formula evaluates to one of $\{\top,\bot\}$, in which case the future is not important.

Now, the request/acknowledge property is evaluated as follows:

$$\begin{aligned}
G(r\rightarrow Fa) &\equiv_{RV} (r\rightarrow Fa)\wedge\bar{X}(G(r\rightarrow Fa))\\
&\equiv_{RV} (\neg r\vee a\vee XFa)\wedge\bar{X}(G(r\rightarrow Fa))
\end{aligned}$$

This formula evaluates to $\bot^p$ under RV-LTL if the trace contains an r but ends before a occurs and evaluates to $\top^p$ in all other cases. Thus, its semantics is exactly as demanded in the beginning of this section.

6 Monitors for RV-LTL

A monitor is a procedure that consumes the input letter by letter and outputs the semantics of the word read so far with respect to the formula the monitor was built for.

In our setting, we use a *Moore machine*, also called *finite-state machine* (FSM), which is a finite state automaton enriched with output. Formally, an FSM is a tuple $\mathcal{A}=(\Sigma,Q,Q_0,\delta,\Delta,\lambda)$, where

– Σ is a *finite alphabet*,
– Q is a finite non-empty set of *states*,
– $q_0\in Q$ is the *initial state*,
– $\delta:Q\times\Sigma\rightarrow Q$ is the *transition function*,
– Δ is the *output alphabet*, and
– $\lambda:Q\rightarrow\Delta$ is the *output function*.

The output of a Moore machine, defined by the function λ, is thus determined by the current state $q\in Q$ alone, rather than by input symbols.

We extend the transition function $\delta:Q\times\Sigma\rightarrow Q$, as usual, to $\delta':Q\times\Sigma^*\rightarrow Q$ with $\delta'(q,\epsilon)=q$ where $q\in Q$ and $\delta'(q,ua)=\delta(\delta'(q,u),a)$. To simplify notation, we use δ for both δ and δ'. Similarly, we extend the output function $\lambda:Q\rightarrow\Delta$ to $\lambda':Q\times\Sigma^*\rightarrow\Delta$ with $\lambda'(q,u)=\lambda(\delta(q,u))$, for $q\in Q$ and $u\in\Sigma^*$. Thus, function λ' yields for a given word u the output in the state reached by u rather than the sequence of outputs. To simplify notation, we use λ for both λ and λ'. We also say that $\mathcal{A}$ *computes* the function $\lambda:\Sigma^*\rightarrow\Delta$.

Following the characterization of RV-LTL in terms of LTL$_3$ and FLTL developed in the previous section, we base the monitor construction for RV-LTL on the monitor constructions for these two incorporated logics.

Monitors for LTL$_3$: In Ref. [2], we presented a monitor construction for a given formula φ to obtain an FSM $\mathcal{A}_3^\varphi$ which computes $[u \models \varphi]_3$ for an incrementally expanded u.

THEOREM 1 [2]
Let φ be an LTL formula. Then there is an effective procedure constructing an FSM $\mathcal{A}_3^\varphi = (\Sigma, Q, q_0, \delta, \mathbb{B}_3, \lambda)$ such that for all $u \in \Sigma^*$ the following holds:

$$\lambda(\delta(q_0, u)) = [u \models \varphi]_3$$

Moreover, the size of $\mathcal{A}_3^\varphi$ is at most double exponential in the size of φ.

Monitors for FLTL: following Ref. [17] and given a formula φ, it is straightforward to come up with a non-deterministic automaton which accepts precisely the words u with $[u \models \varphi]_F = \top$. Such an automaton can be made deterministic with the power-set method [11]. Finally, the deterministic automaton yields an FSM by outputting $\top$ in each accepting state and $\bot$ in all remaining states. Since FLTL is only defined for non-empty words (Remark 4), we have to handle the empty word as special case—and hence we arrive at the following theorem:

THEOREM 2 [17]
Let φ be an LTL formula. Then there is an effective procedure constructing an FSM $\mathcal{A}_F^\varphi = (\Sigma, Q, q_0, \delta, \mathbb{B}, \lambda)$ such that for all $u \in \Sigma^*$, the following holds:

$$\lambda(\delta(q_0, u)) = \begin{cases} [u \models \varphi]_F & \text{for } u \neq \epsilon \\ \top & \text{for } u = \epsilon \end{cases}$$

Moreover, the size of $\mathcal{A}_F^\varphi$ is at most double exponential in the size of φ.

Monitors for RV-LTL: we are now ready to define the monitor $\mathcal{A}_{RV}^\varphi$ computing the RV-LTL semantics by incorporating the respective monitor procedures $\mathcal{A}_3^\varphi$ and $\mathcal{A}_F^\varphi$ for LTL$_3$ and FLTL.

DEFINITION 10 (Monitor $\mathcal{A}_{RV}^\varphi$ for an RV-LTL-formula φ)
Let φ be an LTL formula with $\mathcal{A}_3^\varphi = (\Sigma, Q, q_0, \delta, \mathbb{B}_3, \lambda)$ as the corresponding LTL$_3$ monitor (as stated in Theorem 1) and $\mathcal{A}_F^\varphi = (\Sigma, Q', q_0', \delta', \mathbb{B}, \lambda')$ as the corresponding FLTL monitor (as stated in Theorem 2).

Then we define the *monitor* $\mathcal{A}_{RV}^\varphi$ as the FSM $(\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \mathbb{B}_4, \bar{\lambda})$, where

- $\bar{Q} = Q \times Q'$,
- $\bar{q}_0 = (q_0, q_0')$,

– $\bar{\delta}((q,q'),a) = (\delta(q,a), \delta'(q',a))$, and
– $\bar{\lambda} : \bar{Q} \to \mathbb{B}_4$ is defined by

$$\bar{\lambda}((q,q')) = \begin{cases} \top & \text{if } \lambda(q) = \top \\ \bot & \text{if } \lambda(q) = \bot \\ \top^p & \text{if } \lambda(q) = ? \text{ and } \lambda'(q') = \top \\ \bot^p & \text{if } \lambda(q) = ? \text{ and } \lambda'(q') = \bot \end{cases}$$

Thus, we simultaneously compute the LTL$_3$ and the FLTL semantics by taking the Cartesian product of their respective monitors. The evaluation computed by the new combined monitor forwards $\top$ and $\bot$ from the LTL$_3$ monitor but replaces every inconclusive verdict (?) of the LTL$_3$ monitor by either presumably true ($\top^p$) or presumably false ($\bot^p$). It chooses $\top^p$ if the FLTL monitor outputs $\top$ as its verdicts and $\bot^p$ otherwise.

Since the FLTL must handle the empty word as special case, the resulting RV-LTL monitor treats the empty word ϵ as special case as well: If the monitored property φ is an LTL-tautology, then we have $[\epsilon \models \varphi]_3 = \top$. In this case the truth value of the LTL$_3$ monitor is forwarded without modification, and $\bar{\lambda}(\bar{\delta}(\bar{q}_0, \epsilon)) = \top$ holds. Likewise, we have $\bar{\lambda}(\bar{\delta}(\bar{q}_0, \epsilon)) = \bot$ whenever $[\epsilon \models \varphi]_3 = \bot$. On the other hand, if $[\epsilon \models \varphi]_3 = ?$ holds, then the verdict of the FLTL monitor is used, i.e. a $\top$ verdict of the FLTL monitor results in $\bar{\lambda}(\bar{\delta}(\bar{q}_0, \epsilon)) = \top^p$ and analogously, $\bot$ results in $\top^p$. In summary, we obtain the following theorem:

THEOREM 3 (Correctness of $\mathcal{A}^{\varphi}_{RV}$)
Let φ be an LTL formula and let $\mathcal{A}^{\varphi}_{RV} = (\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \mathbb{B}_4, \bar{\lambda})$ be the monitor according to Definition 10. Then for all $u \in \Sigma^*$, the following holds:

$$\bar{\lambda}(\bar{\delta}(\bar{q}_0, u)) = \begin{cases} [u \models \varphi]_{RV} & \text{for } u \neq \epsilon \\ \top^p & \text{for } u = \epsilon \text{ and } \varphi \not\equiv_\omega \textit{true}, \textit{false} \\ \top & \text{for } u = \epsilon \text{ and } \varphi \equiv_\omega \textit{true} \\ \bot & \text{for } u = \epsilon \text{ and } \varphi \equiv_\omega \textit{false} \end{cases}$$

Moreover, the size of $\mathcal{A}^{\varphi}_{RV}$ is at most double exponential in the size of φ.

The size of the final FSM is in $O(2^{2^n})$ but can be minimized with standard algorithms for FSMs [11] to derive an *optimal* deterministic monitor with a minimal number of states. In the worst case, however, a lower bound of $O(2^{2^{\Omega(n)}})$ applies to the number of states, as follows from [16]. Thus, better complexity results in other approaches, like the one in [14], are due to one of the following reasons:

– first, one can use a fragment of LTL which is *strictly less expressive* than full LTL, i.e. one gives up the possibility to specify certain properties and thereby rules out some complicated cases exercising the worst case complexity. Note that our construction yields an optimal monitor regardless whatever fragment of LTL is considered.
– Second, it is possible to use a variant of LTL which is still capable to express all LTL-expressible properties but which requires *strictly longer formulae* for some of these properties.
– Third, one could abandon a single monolithic and deterministic automaton as monitor procedure, and use instead an alternative concept such as synchronizing automata, hereby trading the size of an automaton with an increased computational overhead at runtime [19].

	LTL	FLTL	LTL$^{\mp}$	LTL$_3$	RV-LTL
Domain	Σ^∞	$u \neq \emptyset, u \in \Sigma^*$(4)	Σ^*(9)	Σ^*(15)	$u \neq \emptyset, u \in \Sigma^*$(21)
Exitential Next (Maxim 1)		yes	yes (+)/no (-)	yes	yes
Complementation by Negation (Maxim 2)		yes	no	no	yes
Impartiality (Maxim 3)		no	no	yes	yes
Anticipation (Maxim 4)		no	no	yes	yes
Boolean laws	yes (1)	yes (6)	no (10)	yes (17)	yes (23)
Equivalences (Fig. 2)	yes (2)	yes (7)	yes (12)	yes (18)	yes (24)
LTL compliant		no (5)	no (11)	yes (16)	no (25)
Negation normalform	yes (3)	yes (8)	yes (13)	yes (19)	yes (24)
Inductive definition	yes	yes	yes	no (14)	no (22)

FIGURE 5. Main properties of the logics studied in this article. The numbers in brackets refer to the remark stating the result

7 Conclusion

In this article we study several variants of LTLs in the context of runtime verification. In runtime verification, we are faced with an incrementally expanding prefix of an unknown infinite trace representing an execution of the underlying system, for which we have to decide whether a property expressed in a LTL holds. Thus, when considering logics for runtime verification, we look for LTLs interpreted over finite traces, with a semantics reflecting that of LTL over infinite traces in a suitable manner.

To this end, we have recalled three existing LTLs interpreted over finite traces, namely, FLTL [17], LTL$^{\mp}$ [8], and LTL$_3$ [2] and elaborated on their properties, for example, which equivalences of formulae hold in the respective logic and how they compare to those in LTL. Moreover, we established four maxims that we consider essential for a LTL aimed for runtime verification:

(1) *Existential next* requires the inclusion of a strong next operator.
(2) *Complementation by negation* requires that a negated formula evaluates to the complemented and different truth value.
(3) *Impartiality* requires that a finite trace is not evaluated to $\top$ ($\bot$) if there still exists an infinite continuation leading to another verdict.
(4) *Anticipation* requires that once every infinite continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this very same verdict.

We analysed FLTL, LTL$^{\mp}$ and LTL$_3$ with respect to these maxims and learnt that none of them satisfies all four of them.

This lead us to the introduction of RV-LTL, whose semantics combines ideas present in LTL$_3$ as well as FLTL. The semantics of RV-LTL indicates whether a finite word describes a system behaviour which either (1) satisfies the monitored property, (2) violates the property, (3) will presumably violate the property or (4) will presumably conform to the property in the future, once the system has stabilized. Using these truth values, we resolved the *ugly* situation of facing an invariably inconclusive verdict in verifying a system at runtime: as long as the final verdict depends on future events, an RV-LTL-based monitor displays a presumably true valuation—if no unanswered request is pending—and presumably false otherwise.

We analysed some basic properties of RV-LTL and especially verified that RV-LTL acts on our four maxims. To turn RV-LTL in a practically applicable device for runtime verification, we developed a monitor generation procedure that relies on corresponding monitor constructions for FLTL and LTL_3. A summarizing comparison of the logics studied in this article is shown in Figure 5.

References

[1] O. Arafat, A. Bauer, M. Leucker, and C. Schallhart. Runtime verification revisited. *Technical Report TUM-I0518*, Technische Universität München, München, 2005.

[2] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *Foundations of Software Technology and Theoretical Computer Science*. S. Arun-Kumar and Naveen Garg, (ed.), Vol. 4337 of *Lecture Notes in Computer Science*, December 2006.

[3] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *Transactions on Software Engineering (TSE)*, 2007. Submitted to ACM Transactions on Software Engineering and Methodology, preliminary version available as *Technical Report TUM-I0724*.

[4] A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly—but how ugly is ugly? In *Proceedings of the 7th International Workshop on Runtime Verification (RV'07)*, Vol. 4839 of *Lecture Notes in Computer Science*, pp. 126–138. Springer-Verlag, Vancouver, Canada, December 2007.

[5] N. D. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-valued Logic*, G. Epstein and J. M. Dunn, eds, pp. 8–37. Reidel, Dordrecht, NL, 1977.

[6] M. d'Amorim and G. Rosu. Efficient monitoring of omega-languages. In *Proceedings of Computer Aided Verification*, K. Etessami and S. K. Rajamani, eds, Vol. 3576 of *Lecture Notes in Computer Science*, pp. 364–378. 2005.

[7] D. Drusinsky. The temporal rover and the ATG rover. In *Proceedings of SPIN Model Checking and Software Verification*, K. Havelund, J. Penix and W. Visser, eds, Vol. 1885 of *Lecture Notes in Computer Science*, Springer, pp. 323–330. 2000.

[8] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *Conference on Computer Aided Verification*. Warren A. Hunt Jr. and Fabio Somenzi, (ed.), Vol. 2725 *of Lecture Notes in Computer Science*, pp. 27–39. 2003.

[9] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *ASE*, M. Feather and M. Goedicke, (ed.), pp. 412–416. IEEE Computer Society, 2001.

[10] D. Giannakopoulou and K. Havelund. Runtime analysis of linear temporal logic specifications. *Technical Report 01.21*, RIACS/USRA, Mountain View, 2001.

[11] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computation*, Z. Kohavi and A. Paz, (eds), New York, Academic, pp. 189–196. 1971.

[12] K. Havelund and G. Rosu. Monitoring Java Programs with Java PathExplorer. *Electronic Notes in Theoretical Computer Science*, **55**, 200–217, 2001.

[13] K. Havelund and G. Rosu. Monitoring programs using rewriting. In *ASE*, M. Feather and M. Goedicke, (ed.), p. 135. IEEE Computer Society, 2001.

[14] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*. J.-P. Katoen and P. Stevens, (ed.), Vol. 2280 *of Lecture Notes in Computer Science*, pp. 342–356. 2002.

[15] H. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[16] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, **19**, 291–314, 2001.

[17] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995.

[18] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science*, pp. 46–57. IEEE Computer Society Press, 31 October to 2 November 1977.

[19] G. Rosu and S. Bensalem. Allen linear (interval) temporal logic - translation to LTL and monitor synthesis. In *Conference on Computer Aided Verification*. Vol. 4144 *of Lecture Notes in Computer Science*, T. Ball and R. B. Jones, (ed.), pp. 263–277. 2006.

[20] V. Stolz and E. Bodden. Temporal assertions using AspectJ. In *RV*. H. Barringer, *et al*. (eds), Vol. 144/4 of *Electronic Notes in Theoretical Computer Science*, 2005.

[21] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27–September 3, 1995*, F. Moller and G. M. Birtwistle, (eds), Vol.1043 of *Lecture Notes in Computer Science*, Springer, 1996.

[22] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Logic in Computer Science*, A. Meyer, (ed.), pp. 332–345. 1986.

Received June 7, 2008