The good, the bad, and the ugly, but how ugly is ugly?

Andreas Bauer[1], Martin Leucker[2], and Christian Schallhart[2]

[1] National ICT Australia (NICTA)
[2] Institut für Informatik, Technische Universität München

Abstract. When monitoring a system wrt. a property defined in some temporal logic, e. g., 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 streams of events, whereas at runtime only prefixes are available.
This work defines a four-valued semantics for LTL over finite traces, which extends the classical semantics, and allows to infer whether a system behaves (1) according to the monitored property, (2) violates the property, (3) will possibly violate the property in the future, or (4) will possibly conform to the property in the future, once the system has stabilised. Notably, (1) and (2) correspond to the classical semantics of LTL, whereas (3) and (4) 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 a Moore machine signalising the semantics of the so far obtained execution trace.

1 Introduction

Runtime verification of a given correctness property φ formulated in linear temporal logic LTL [Pnu77] aims at determining the semantics of φ while executing the system under scrutiny. However, one is faced with the following obstacle: The semantics of LTL is defined over infinite (behavioural) traces whereas monitoring a running system allows an at most finite view.

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 strings. Several versions of a *two-valued* semantics for LTL on finite strings have been proposed [GH01a,HR01b,HR02,HR01a,SB05,dR05], see Eisner et al. for a comprehensive survey on this topic [EFH$^+$03]. Alternatively, it has also 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 [GH01b].

In monitoring a property, there can at least arise three different situations: Firstly, the property can be already satisfied for sure after a finite number of steps; secondly, the property can be shown to evaluate to false for every possible continuation, or thirdly, the finite, already observed prefix allows different continuations leading to either satisfaction or falsification. Thus, every two-valued logic must evaluate to true or false prematurely since it cannot reflect the third case properly.

To overcome these obstacles, we propose in [ABLS05,BLS06], a three-valued semantics which extends the classical semantics over finite traces. There, a property evaluates to *true* (*false*), wrt. a finite observation, iff the observation is either a satisfying (violating) prefix. In all other cases, the observation is said to be inconclusive, and the property assigned a ?.

This scheme coincides well with the notion of *safety* (e. g., Gp—always p) and *co-safety* (e. g., Fp—eventually p) properties, since these are either finitely refutable or satisfiable. However, when monitoring a *true liveness* property, that is one that is not safety nor a co-safety property, then neither the violation nor the satisfaction of the property can be determined using a finite stream of observations, and not much is said about the possible future. Actually, in [PZ06] it is suggested to call these properties *non-monitorable*.

A typical example for a liveness property is $G(request \rightarrow F\,grant)$ saying that every request should eventually be granted. In practice, however, one is often faced with such properties and therefore it is impractical to preclude corresponding monitoring procedures.

In this work, we follow the idea that an inconclusive result of a monitor should be more detailed. To this end, we propose a four-valued semantics for LTL that not only results in either *true*, *false*, or ?, but yields *possibly true* and *possibly false* whenever the system's behaviour so far is inconclusive in the three-valued sense. We call the resulting logic *Runtime Verification Linear Temporal Logic* (RV-LTL).

Further, we have defined a translation from a formula in RV-LTL to a monitor (Moore machine) of minimal size, which then forms a suitable foundation for runtime verification, in that the output alphabet of the automaton corresponds to the four truth values sketched above.

Our logic RV-LTL seems to correspond with the semantics realised by the Temporal Rover [Dru00] and has, to the best of our knowledge, not been formally captured elsewhere.

In [dR05], a monitor construction and simplification is given. By combining two of their monitors, as briefly described in their implementation section, this approach can be used to implement the three-valued semantics as presented independently in [ABLS05,BLS06].

Outline. We briefly recall the standard infinite trace semantics of LTL in the next section. In Section 3, we first elaborate four maxims which we require to be satisfied by a temporal logic suitable for runtime verification. Then, we recall two preexisting logics for finite traces and show why they do not satisfy our maxims. However, we show how to combine the two logics towards RV-LTL and argue that RV-LTL adheres to our maxims. Finally, in Section 4, we describe a construction of a monitor procedure for RV-LTL.

Acknowledgement. We thank the anonymous reviewers for their valuable and detailed comments.

2 LTL on infinite traces

For the remainder of this paper, let AP be a finite 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 set of propositions taken from AP.

Finite traces over Σ are elements of Σ^*, usually denoted by $u, u', u_1, u_2, \ldots$, whereas infinite traces are elements of Σ^ω, usually denoted by $w, w', w_1, w_2, \ldots$. For some trace $w = a_0 a_1 \ldots$, we denote by w^i the suffix $a_i a_{i+1} \ldots$.

The set of LTL formulae is inductively defined by the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\ U\ \varphi \mid X\varphi \quad (p \in \mathrm{AP}) \tag{1}$$

Let $i \in \mathbb{N}$ be a *position*. The semantics of LTL formulae is defined inductively over infinite sequences $w = a_0 a_1 \ldots \in \Sigma^\omega$ as follows:

$$\begin{array}{lll}
w, i \models \mathit{true} & & \\
w, i \models \neg\varphi & \text{iff} & w, i \not\models \varphi \\
w, i \models p & \text{iff} & p \in a_i \\
w, i \models \varphi_1 \vee \varphi_2 & \text{iff} & w, i \models \varphi_1 \text{ or } w, i \models \varphi_2 \\
w, i \models \varphi_1 U \varphi_2 & \text{iff} & \text{there is a } k \geq i : w, k \models \varphi_2 \text{ and} \\
 & & \quad \text{for all } l \text{ with } i \leq l < k : w, l \models \varphi_1 \\
w, i \models X\varphi & \text{iff} & w, i+1 \models \varphi
\end{array}$$

Further, let $w \models \varphi$ be an abbreviation for $w, 0 \models \varphi$. We call w a *model* of φ iff $w \models \varphi$. For every LTL formula φ, its set of models, denoted by $\mathcal{L}(\varphi)$, is a regular set of infinite traces and can be described by a corresponding Büchi automaton [VW86,Var96]. For $\varphi, \psi \in$ LTL, we say that φ is equivalent to ψ, denoted by $\varphi \equiv \psi$, iff for all $w \in \Sigma^\omega$, we have

$$w \models \varphi \text{ iff } w \models \psi$$

For reasons to become clear in Section 3, note that, in LTL

$$\neg X\varphi \equiv X\neg\varphi \tag{2}$$

holds, which matches the intuition that something does not hold in the next position if, in the next position, it does not hold.

3 LTL on finite traces

While the syntax and semantics of LTL on infinite traces is well-accepted in the literature, there is no consensus on defining LTL on finite strings. Several versions of a *two-valued* semantics for LTL on finite strings have been proposed. However, as we argue below, for runtime verification, a *four-valued* semantics is preferable.

As discussed in [MP95], the difficulty for an LTL semantics on finite strings lies in the next-state operator X. Given a finite string $u = a_0, \ldots, a_{n-1}$ of length n, the question is which semantics to choose for $X\varphi$ in the last position of u:

$$u, n-1 \overset{?}{\models} X\varphi \tag{3}$$

We follow the approach of [MP95] in understanding the next-state operator as an operator firstly assuring that there exists a next state which secondly satisfies φ. We use this assumption as the first of our four maxims, which we consider essential for a semantics for LTL on finite traces, in particular in the context of runtime verification:

- *$X\varphi$ means there exists a next state and this state satisfies φ.* **($\exists$X)**

Consequently, equation 3 yields false, as there is no next state. Our second maxim states that a negated formula indeed yields the complemented truth value of the original formula, i. e.,

- *a formula and its negation yield complementary truth values.* **($\neg$=C)**

Then, however, a negated next-state formula should be true. This, however, conflicts equation 2 ($\neg X\varphi \equiv X\neg\varphi$), which therefore can no longer hold on finite traces (unless true equals false). It is therefore helpful to distinguish a *strong* (denoted by X) and a *weak* version (denoted by $\bar{X}$) of the next-state operator.

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 $\bar{X}$ also *universal* next-state operator.

The introduction of a strong and a weak version of a 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 [HR02], 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.

In LTL, we have that $F\varphi \equiv \varphi \vee XF\varphi$, 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 $\bar{X}$. Using this notation, we can rewrite the above LTL equivalences as $F\varphi \equiv \varphi \vee XF\varphi$ and as $G\varphi \equiv \varphi \wedge \bar{X}G\varphi$.

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 logic suitable for runtime verification. The first one says that

– *the semantics never evaluates to true or false prematurely.* **(Sound)**

The string a (of length 1) clearly satisfies the proposition p iff $p \in a$. While, understanding a as a prefix of an infinite string, the value of $X\varphi$ is of less certainty, as the successor state of a is not known. Choosing either true or false (depending on whether to understand X strongly or weakly) would diminish the qualitative difference of the knowledge on p and $X\varphi$ based on the string a. Therefore, we require a semantics to yield four values: *true*, *possibly true*, *possibly false*, and *false*.

When considering $X\varphi$ in the last state of a finite string u, there is no reason to evaluate $X\varphi$ to false, possibly false, or possibly true, if every possible continuation of u satisfies φ. A trivial example would be $X\,\mathit{true}$. While every single letter extension of u would make $X\,\mathit{true}$ true in u's last position, the semantics discussed so far evaluate $X\,\mathit{true}$ to false or possibly false. We therefore postulate

– *the semantics is as anticipatory as possible.* **(Precise)**

In the remainder of the section, we recall two preexisting logics for finite traces, namely FLTL and LTL_3, and show that they do not satisfy all our postulated maxims **(∃X)**, **(¬=C)**, **(Sound)**, and **(Precise)**. Then, we combine the two logics towards RV-LTL and argue that RV-LTL adheres to our maxims.

3.1 Existing Semantics for Finite Traces

We start by recalling two existing definitions of LTL for finite traces, namely FLTL [LPZ85] and LTL_3 [BLS06]. Both variants provide complementary properties for runtime verification but neither of them satisfies all four maxims as postulated above.

The set of FLTL formulae is inductively defined by the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\ U\ \varphi \mid X\varphi \mid \bar{X}\varphi \quad (p \in \mathrm{AP}) \tag{4}$$

In this definition, we use two versions of the next-state operator to overcome the difficulty of deciding whether a formula $X\varphi$ holds in the last position of a finite string—thus FLTL satisfies **(∃X)**: The *strong* (and standard) 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.

The semantics function $[u, i \models \varphi]_F$ of FLTL is constructed like the one for standard LTL with one modification: 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 concept is explicated in the following definition:

Definition 1 (Semantics of FLTL [LPZ85]). *Let $u = a_0 \ldots a_{n-1} \in \Sigma^*$ denote a finite trace of length n. The* truth value *of an FLTL formula φ wrt. u*

at position $i < n$, denoted by $[u, i \models \varphi]_F$, is an element of $\mathbb{B}$ and is defined as follows:

$$
\begin{aligned}
[u, i \models true]_F &= \top \\
[u, i \models p]_F &= \begin{cases} \top & \text{if } p \in a_i \\ \bot & \text{if } p \notin a_i \end{cases} \\
[u, i \models \neg\varphi]_F &= \overline{[u, i \models \varphi]_F} \\
[u, i \models \varphi \vee \psi]_F &= [u, i \models \varphi]_F \sqcup [u, i \models \psi]_F \\
[u, i \models X\varphi]_F &= \begin{cases} [u, i+1 \models \varphi]_F & \text{if } i+1 < n \\ \bot & \text{otherwise} \end{cases} \\
[u, i \models \bar{X}\varphi]_F &= \begin{cases} [u, i+1 \models \varphi]_F & \text{if } i+1 < n \\ \top & \text{otherwise} \end{cases} \\
[u, i \models \varphi\ U\ \psi]_F &= [u, i \models \psi]_F \sqcup ([u, i \models \varphi]_F \sqcap [u, i \models X\ U\ \psi]_F
\end{aligned}
$$

Therefore, FLTL can satisfy the maxim **($\exists$X)** as well as **($\neg$=C)** since negated formulae always yield complementary truth values. However, FLTL does not satisfy the maxim **(Precise)** because the truth value of $[u, n-1 \models X\varphi]_F$ for $|u| = n$ does not depend on φ at all. For example, $[u, n-1 \models Xtrue]_F = \bot$ although $Xtrue$ will evaluate to $\top$ in every possible continuation. Furthermore, FLTL cannot satisfy maxim **(Sound)** since FLTL only uses a two-valued semantic domain and thus every prefix must be evaluated (possibly prematurely) to either $\top$ or $\bot$.

In[BLS06,ABLS05], we proposed LTL$_3$ as an LTL logic with a semantics for finite traces, which caters the view that a finite trace is a prefix of an so-far unknown infinite trace. More specifically, LTL$_3$ uses the standard syntax of LTL as defined in Equation (1) but employs a semantics function $[u, i \models \varphi]_3$ which evaluates for a formula φ each finite trace u of length n and each position $0 \leq i < n$ to a value out of $\mathbb{B}_3 = \{\top, \bot, ?\}$. If every infinite trace with prefix u evaluates to same truth value $\top$ or $\bot$, then $[u, i \models \varphi]_3$ also evaluates to this truth value. Otherwise $[u, i \models \varphi]_3$ evaluates to ?, i. e., we have $[u, i \models \varphi]_3 =?$ if different continuations of u yield different truth values. This discussion leads to the following definition:

Definition 2 (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 wrt. u at position $i < n$, denoted by $[u, i \models \varphi]_3$, is an element of $\mathbb{B}_3$ and defined as follows:*

$$
[u, i \models \varphi]_3 = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma, i \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma, i \not\models \varphi \\ ? & \text{otherwise.} \end{cases}
$$

Note that LTL$_3$ satisfies three of our four maxims: A formula and its negation yield the complementary truth values **($\neg$=C)**, the semantics never evaluates to true or false prematurely **(Sound)**, and the semantics are as anticipatory as possible **(Precise)**. Since LTL$_3$ uses the standard LTL syntax, it does not distinguish between a strong and weak next-state operator and consequently, the maxim **($\exists$X)** cannot be satisfied.

Note that ideas leading to the definition of LTL_3 have been formulated independently in [dR05]. The notion of a *(minimal) bad prefix* u is introduced and defined as a prefix that does not have any continuation satisfying a formula φ. Thus, $[u \models \varphi]_3$ would evaluate to $\bot$ for every bad prefix u for φ. By adding a dual monitor for $\neg\varphi$, as proposed in the implementation section of [dR05], the notion of a *good prefix* is obtained and the semantics of LTL_3 can be implemented.

3.2 RV-LTL

We now define RV-LTL which is a version of LTL on finite strings tailored for runtime verification. RV-LTL is designed to incorporate and resolve the requirements as stated afore. The set of RV-LTL formulae is inductively defined by the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\ U\ \varphi \mid X\varphi \mid \bar{X}\varphi \quad (p \in \mathrm{AP}) \tag{5}$$

As in FLTL, we use two versions of the next-state operator to overcome the difficulty of deciding whether a formula $X\varphi$ holds in the last position of a finite string—thus RV-LTL satisfies **($\exists$X)**. Like in FLTL, the *strong* (and standard) X operator is used to express with $X\varphi$ that a next state must exist and that his next state has to satisfy some property φ. Dually, the *weak* $\bar{X}$ operator in $\bar{X}\varphi$ says that if there is a next state, then this next state must satisfy the property φ.

To accommodate maxim **(Sound)** and in contrast to FLTL, 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. $\mathbb{B}_4$ can be extended to a complete lattice by ordering $\bot \leq \bot^p \leq \top^p \leq \top$. $\sqcap$ and $\sqcup$ are then defined as expected. To match maxim **($\neg$=C)**, $\bot$ and $\top$ are defined to be complementary to each other as well as $\bot^p$ and $\top^p$, where complementation is denoted by $\bar{\ }$. Note that $\mathbb{B}_4$ is not a Boolean lattice, as, for example, $\bot^p \sqcup \overline{\bot^p} = \bot^p \sqcup \top^p \neq \top$. However, the distributive laws hold:

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

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

Definition 3 (Semantics of RV-LTL). *Let $u = a_0 \ldots a_{n-1} \in \Sigma^*$ denote a finite trace of length $n = |u|$. The* truth value *of an RV-LTL formula φ wrt. u at position $i < n$, denoted by $[u, i \models \varphi]_{RV}$, is an element of $\mathbb{B}_4$ and is defined as follows:*

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

where φ' is obtained from φ by replacing each weak next-state operator $\bar{X}$ with a strong next-state operator X.

Note that, in the last position of a word u, both $X\varphi$ and $\bar{X}\varphi$ evaluate to $\top$ ($\bot$) if the outcome is predetermined for all possible continuations. Therefore, the semantics of RV-LTL also satisfy maxim **(Precise)**.

Note that the semantics of RV-LTL as given in Definition 3 directly provides an efficient way to construct a monitor procedure for RV-LTL: By running a monitor for LTL_3 and for FLTL simultaneously and by combining their respective results following Definition 3, we obtain a monitor procedure for RV-LTL. We will exploit this fact in the next section where we discuss the monitor construction for RV-LTL in detail.

For two formulae φ and ψ, we say that φ is equivalent to ψ w.r.t. RV-LTL, denoted by $\varphi \equiv_{RV} \psi$, iff for all $u \in \Sigma^*$ and $0 \leq i < |u|$, we have $[u, i \models \varphi]_{RV} = [u, i \models \psi]_{RV}$.

To demonstrate the semantics of RV-LTL, we discuss in the following a number of examples. In the motivating discussion of this section, we referred to the equivalence

$$\neg X\varphi \equiv X\neg\varphi$$

which is true w.r.t. LTL. On the other hand, in RV-LTL, this equivalence does not hold, as $\neg X\varphi$ is $\top^p$ in the last position of a word u (if φ cannot be evaluated for possible continuations), while $X\neg\varphi$ is $\bot^p$. However, we have

$$\neg X\varphi \equiv_{RV} \bar{X}\neg\varphi.$$

Using the equivalence

$$\varphi\ U\ \psi \equiv_{RV} \psi \vee (\varphi \wedge X(\varphi\ U\ \psi))$$

which holds for RV-LTL as well as for standard LTL, we can define the finally operator F and globally operator G as abbreviations $F\varphi := \mathit{true}\ U\ \varphi$ and $G\varphi := \neg F\neg\varphi$ and evaluate them as follows:

$$\begin{aligned} F\varphi &\equiv_{RV} \mathit{true}\ U\ \varphi \\ &\equiv_{RV} \varphi \vee (\mathit{true} \wedge X(\mathit{true}\ U\ \varphi)) \\ &\equiv_{RV} \varphi \vee XF\varphi \end{aligned}$$

and

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

yield the two equivalences $F\varphi \equiv_{RV} \varphi \vee XF\varphi$ and $G\varphi \equiv_{RV} \varphi \wedge \bar{X}\,G\varphi$ which we discussed to motivate our four maxims. Note that in the previous calculation we used the distributive law and the equivalence $\neg X\varphi \equiv_{RV} \bar{X}\neg\varphi$.

$F\varphi \equiv_{RV} \varphi \vee XF\varphi$ 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 $\bot^p$. 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 $\top^p$.

As a final example, we evaluate the property that some request must be answered by a corresponding answer:

$$\begin{aligned} G(p \rightarrow Fq) &\equiv_{RV} (p \rightarrow Fq) \wedge \bar{X}(G(p \rightarrow Fq)) \\ &\equiv_{RV} (\neg p \vee q \vee XFq) \wedge \bar{X}(G(p \rightarrow Fq)) \end{aligned}$$

This formula evaluates to $\bot^p$ under RV-LTL if the trace contains a p but ends before q occurs and evaluates to $\top^p$ in all other cases. This behaviour is intuitive, since the first case corresponds to a request which has not been answered yet, while the second case means that all requests so far have been answered properly.

Let us close this section by recalling that RV-LTL's semantics can be understood as refinement of LTL_3's semantics. Consequently, we can obtain the semantics of LTL_3 by mapping a $\top^p/\bot^p$ value to ?:

Remark 1. Let $u = a_0 \ldots a_{n-1} \in \Sigma^*$ denote a finite trace of length n and let φ be an LTL_3 formula. Then the following holds

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

where the X of LTL_3 is interpreted as strong next-state operator in RV-LTL.

4 Monitors for RV-LTL

A monitor is a device that consumes the input letter by letter and outputs the semantics of the string 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 $\delta : Q \times \Sigma \to Q$, as usual, to $\delta' : Q \times \Sigma^* \to Q$ by $\delta'(q, \epsilon) = q$ 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 $\lambda : Q \to \Delta$ to $\lambda' : Q \times \Sigma^* \to \Delta$ by $\lambda'(q, u) = \lambda(\delta(q, u))$, 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 $\lambda : \Sigma^* \to \Delta$.

Following the characterisation 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 the respective logics.

Monitors for LTL_3. In [BLS06], a monitor construction for a given formula φ with respect to the three-valued semantics was elaborated:

Theorem 1 ([BLS06]). *Let φ be an LTL_3 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:*

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

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

Monitors for FLTL. Following [MP95], it is easy to come up with a non-deterministic automaton accepting precisely the words satisfying a given LTL formula φ with respect to the FLTL semantics. Such an automaton can be made deterministic as usual. Moreover, a deterministic automaton can be understood as an FSM by outputting $\top$ in each accepting state and $\bot$ in the remaining states. This gives:

Theorem 2 ([MP95]). *Let φ be an FLTL 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:*

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

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 a monitor computing the RV-LTL semantics.

Definition 4 (Monitor $\bar{\mathcal{A}}_{RV}^\varphi$ for a RV-LTL-formula φ). *Let φ be an RV-LTL formula. Let $\mathcal{A}_3^\varphi = (\Sigma, Q, q_0, \delta, \mathbb{B}_3, \lambda)$ be the monitor that computes the 3-valued semantics for φ (cf. Theorem 1) where weak next has been replaced with the standard next-state operator.*

Moreover, let $\mathcal{A}_F^\varphi = (\Sigma, Q', q_0', \delta', \mathbb{B}, \lambda')$ be the monitor that computes the (two-valued) FLTL semantics of φ (cf. Theorem 2). Then we define the monitor *$\bar{\mathcal{A}}_{RV}^\varphi$ 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 three-valued as well as the FLTL semantics by taking the Cartesian product of the corresponding monitors. However, we keep $\top$ and $\bot$ from the three-valued semantics and go for possibly true ($\top^p$) or possibly false ($\bot^p$) whenever the three-valued semantics gives don't know (?) and FLTL semantics yields $\top$ or $\bot$, respectively. This gives:

Theorem 3. *Let φ be an RV-LTL formula and let $\bar{\mathcal{A}}^{\varphi}_{RV} = (\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \mathbb{B}_4, \bar{\lambda})$ be the monitor according to Definition 4. Then for all $u \in \Sigma^*$ the following holds:*

$$[u \models \varphi]_{RV} = \bar{\lambda}(\bar{\delta}(\bar{q}_0, u)).$$

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

While the size of the final FSM is in $O(2^{2^n})$ which sounds a lot, standard minimisation algorithms for FSMs can be used to derive an *optimal* deterministic monitor wrt. the number of states. Optimality implies that any other method, in the worst case, has the same complexity. Better complexity results in other approaches are either due to using a restricted fragment of LTL or otherwise imply that the chosen temporal operators might not limit the expressive power of LTL but sometimes impose long formulas for encoding the desired behaviour.

In practice, however, one might trade a precomputed deterministic monitor towards an on-the-fly determinisation on a non-deterministic monitor as described in [BLS06].

5 Conclusion

In this paper we introduced RV-LTL which is a new variant of LTL defined over finite traces. We developed RV-LTL in order to match four maxims which are motivated by runtime verification applications: A suitable semantics for runtime verification should evaluate each formula and its negation to complementary truth values **(¬=C)**, the semantics should never evaluate to true or false prematurely **(Sound)**, the semantics should be as anticipatory as possible **(Precise)**, and finally, the logic should provide a strong and weak next-state operator **(∃X)**. While preexisting logics can satisfy these maxims partially, none of them does satisfy all four properties simultaneously.

This gap is closed by RV-LTL which matches all four maxims. To turn RV-LTL in a practically applicable device for runtime verification, we first showed how to define the semantics of RV-LTL in terms of two other variants of LTL, namely LTL_3 and FLTL, and second we translated this relationship into an efficient monitor construction.

References

[ABLS05] Oliver Arafat, Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification revisited. Technical Report TUM-I0518, Technische Universität München, 2005.

[BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *LNCS*. Springer, December 2006.

[dR05] Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of omega-languages. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 364–378. Springer, 2005.

[Dru00] Doron Drusinsky. The temporal rover and the atg rover. In *SPIN*, volume 1885 of *LNCS*, pages 323–330. Springer, 2000.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *CAV03*, volume 2725 of *LNCS*, pages 27–39, Boulder, CO, USA, July 2003. Springer.

[GH01a] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *ASE*, pages 412–416. IEEE Computer Society, 2001.

[GH01b] Dimitra Giannakopoulou and Klaus Havelund. Runtime analysis of linear temporal logic specifications. Technical Report 01.21, RIACS/USRA, 2001.

[HR01a] Klaus Havelund and Grigore Rosu. Monitoring Java Programs with Java PathExplorer. *Electr. Notes Theor. Comp. Sci.*, 55(2), 2001.

[HR01b] Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *ASE '01: Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, page 135, Washington, DC, USA, 2001. IEEE Computer Society.

[HR02] Klaus Havelund and Grigore Rosu. Synthesizing Monitors for Safety Properties. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 342–356, 2002.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The Glory of the Past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, 1985.

[MP95] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, 1977. IEEE.

[PZ06] Amir Pnueli and Aleksandr Zaks. Psl model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM*, volume 4085 of *LNCS*, pages 573–586. Springer, 2006.

[SB05] Volker Stolz and Eric Bodden. Temporal Assertions using AspectJ. In *Fifth Workshop on Runtime Verification (RV'05)*. To be published in ENTCS, Elsevier, 2005.

[Var96] Moshe Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS'86)*, pages 332–345, Washington, D.C., USA, June 1986. IEEE Computer Society Press.