Sliding between model checking and runtime verification

Martin Leucker

Universtität zu Lübeck
Institut für Softwaretechnik und Programmiersprachen

Abstract. We present a unified semantics for linear temporal logic capturing model checking and runtime verification. Moreover, we present the main ingredients of a corresponding monitor synthesis procedure.

1 Introduction

One of the main research problems in runtime verification (RV) is the automatic synthesis of monitors from high-level specifications. A typical high-level specification language used in RV is linear temporal logic (LTL), for which different semantical, RV-specific adaptions have been proposed in recent years. In this paper we propose a further semantics for LTL, a *predictive semantics*, which unifies ideas from model checking and runtime verification. Using abstraction, it allows to either concentrate on a more model checking like analysis of the underlying system, or a rather runtime verification oriented view on the system under scrutiny.

The main object to study in RV is the current *run* of the underlying system. Such a run may be finite or, at least from a theoretical point of view, infinite, for example when the execution of the reactive system like a web server is considered. However, when the underlying system runs, we can only observe a finite part of a potentially infinite run. We call the finite, observed part of the run an *execution*. Thus, in RV, we observe executions of an underlying system and want to assess the correctness of a high-level specification with respect to the run of the system. Now, one can come up with different semantics for LTL depending on how we understand executions and runs in detail.

In the basic case an execution and run coincide. Here, we think of a system that executed for a finite amount of time and the execution has terminated. This is the case for example when analyzing log files or when dealing with classical input/output oriented computations. If we want to analyze a correctness property for such an execution, an LTL semantics on finite words is most appropriate. A bunch of different variations of LTL semantics on finite words have been proposed in the literature, implicitly in Kamp's work [1] and more directly in Manna and Pnueli's work [2] or more recently by Eisner et al. in the context of LTL^+ and LTL^- [3] (see [4] for a comparison).

For reactive systems, however, the typical view on a computation is no longer the input/output behavior but the interaction of the system with its environment. In an ideal case such a run is infinite. For example, in the setting of a

web server we are not interested in a kind of final result of the server but deal with questions such as whether the web server follows the underlying protocol. Here, we consider an execution to be a *prefix of a potentially infinite run*. An appropriate semantics for such a setting with respect to RV was given in [5].

The idea is that a correctness property is evaluated on the current execution u with respect to all possible further extensions of the current execution. The rational is that it is fair to evaluate u with respect to all possible extensions as we know that u will extend somehow, but we do not really know how. If u together with all possible extensions satisfies the correctness property the runtime verification semantics of u with respect to the property is *true*. When all extensions of u violate the given correctness property the RV semantics of u with respect to the correctness property yields *false* while in all other cases the RV semantics yields ? meaning that no conclusive answer could be given. In other words, we give a three-valued semantics to LTL properties based on all possible extensions of the current execution.

In this paper we build on the previous idea, however, we extend the approach towards a *predictive* semantics by the following observations. Why do we check all possible extensions of the current execution u? Given a program $\mathcal{P}$ it seems to be more interesting to consider only the possible executions of the program $\mathcal{P}$. If we follow this idea we get *true* and *false* for the underlying property in more cases. In a sense, such a semantics would be more *precise*.

However, consider RV with such an idea right at the start for the empty word ϵ. We then have to check whether all the extensions of the empty word following the program $\mathcal{P}$ would satisfy our correctness property. Thus, we check whether all runs of our program $\mathcal{P}$ satisfy our correctness property and hence answer the model checking question. In consequence, we have to deal with the so-called state-space explosion also in RV.

The situation changes when we look at an abstraction $\hat{\mathcal{P}}$ of the underlying program $\mathcal{P}$ that has more runs than the original program $\mathcal{P}$. Then we can look at all extensions of an execution u with respect to the abstract system $\hat{\mathcal{P}}$. This may yield a more precise assessment than the original three-valued semantics but may be easier to check than model checking. Moreover depending on the level of abstraction one can focus more on the runtime verification aspects or more on the model checking ideas. In one of the extreme cases $\hat{\mathcal{P}}$ and $\mathcal{P}$ coincide and we solve the model checking problem while in the other extreme case $\hat{\mathcal{P}}$ just contains all possible executions over a given alphabet and we are in the traditional setting of three-valued LTL.

In this paper we show that $\hat{\mathcal{P}}$ can actually be combined with a previous monitor synthesis procedure for three-valued LTL so that a monitor for the resulting predictive semantics is obtained. More precisely, the resulting monitor checks the semantics for a given execution u and a correctness property with respect to an abstraction $\hat{\mathcal{P}}$ of the underlying program $\mathcal{P}$.

In the remainder of this paper we make the previous ideas precise.

2 Preliminaries

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. Finite traces over Σ are elements of Σ^*, and are usually denoted by $u, u', u_1, u_2, \ldots$, whereas infinite traces are elements of Σ^ω, usually denoted by $w, w', w_1, w_2, \ldots$.

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

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

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: $w, i \models \mathit{true}$, $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$, $w, i \models p$ iff $p \in a_i$, $w, i \models \varphi_1 \vee \varphi_2$ iff $w, i \models \varphi_1$ or $w, i \models \varphi_2$, $w, i \models \varphi_1 U \varphi_2$ iff there exists $k \geq i$ with $w, k \models \varphi_2$ and for all l with $i \leq l < k$, $w, l \models \varphi_1$, and $w, i \models X\varphi$ iff $w, i+1 \models \varphi$. Further, let $w \models \varphi$, iff $w, 0 \models \varphi$. 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.

A (nondeterministic) Büchi automaton (NBA) is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, where Σ is a finite alphabet, Q is a finite non-empty set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is a set of accepting states. We extend the transition function $\delta : Q \times \Sigma \to 2^Q$ to sets of states and (input) words as usual. A *run* of an automaton $\mathcal{A}$ on a word $w = a_1 \ldots \in \Sigma^\omega$ is a sequence of states and actions $\rho = q_0 a_1 q_1 \ldots$, where q_0 is an initial state of $\mathcal{A}$ and for all $i \in \mathbb{N}$ we have $q_{i+1} \in \delta(q_i, a_i)$. For a run ρ, let $\mathrm{Inf}(\rho)$ denote the states visited infinitely often. ρ is called *accepting* iff $\mathrm{Inf}(\rho) \cap F \neq \emptyset$.

A nondeterministic *finite automaton* (NFA) $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, where Σ, Q, Q_0, δ, and F are defined as for a Büchi automaton, operates on finite words. A *run* of $\mathcal{A}$ on a word $u = a_1 \ldots a_n \in \Sigma^*$ is a sequence of states and actions $\rho = q_0 a_1 q_1 \ldots q_n$, where q_0 is an initial state of $\mathcal{A}$ and for all $i \in \mathbb{N}$ we have $q_{i+1} \in \delta(q_i, a_i)$. The run is called accepting if $q_n \in F$. A NFA is called *deterministic* and denoted DFA, iff for all $q \in Q$, $a \in \Sigma$, $|\delta(q, a)| = 1$, and $|Q_0| = 1$.

As usual, the language accepted by an automaton (NBA/NFA/DFA), denoted by $\mathcal{L}(\mathcal{A})$, is given by its set of accepted words.

Let us also recall the notion of a *Moore machine* (also *finite-state machine*, FSM), which is a finite state automaton enriched with output, formally denoted by a tuple $(\Sigma, Q, Q_0, \delta, \Delta, \lambda)$, where Σ, Q, $Q_0 \subseteq Q$, δ is as before and Δ is the output alphabet, $\lambda : Q \to \Delta$ the output function. The outputs of a Moore machine, defined by the function λ, are thus determined by the current state $q \in Q$ alone. As before, δ extends to the domain of words as expected. Moreover, we denote by λ also the function that applied to a word u yields the output in the state reached by u rather than the sequence of outputs.

In this paper, a (finite-state) *program* is given as a non-deterministic Büchi automaton for which all states are final. Runs of a program coincide with the runs of the Büchi automaton. The product of a program $\mathcal{P} = (\Sigma, Q, Q_0, \delta, Q)$ and an NBA $\mathcal{A} = (\Sigma, Q', Q_0', \delta', F')$ is the NBA $\mathcal{B} = (\Sigma, Q \times Q', Q_0 \times Q_0', \delta'', Q \times F')$ where $\delta''((q, q'), a) = \delta(q, a) \times \delta'(q', a)$, for all $q \in Q$, $q' \in Q'$ and $a \in \Sigma$. *Model checking* answers the question whether for a given program $\mathcal{P}$ and an LTL property φ, $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi)$.

3 A predictive semantics for LTL

Let us recall our 3-valued semantics, denoted by LTL_3, over the set of truth values $\mathbb{B}_3 = \{\bot, ?, \top\}$ from [5]: Let $u \in \Sigma^*$ denote a finite trace. The *truth value* of a LTL_3 formula φ wrt. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:

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

In the previous definition, one might ask why not only consider extensions of u that altogether yield runs of an underlying program $\mathcal{P}$. Thus, one might be tempted to define a *predictive* semantics for a finite word u and a property φ with respect to a program $\mathcal{P}$, for example for the case $\top$, by $[u \models \varphi]_{\mathcal{P}} = \top$ iff $\forall \sigma \in \Sigma^\omega$ with $u\sigma \in \mathcal{P} : u\sigma \models \varphi$. However, for the empty word this means $[\epsilon \models \varphi]_{\mathcal{P}} = \top$ iff $\forall \sigma \in \Sigma^\omega$ with $\epsilon\sigma \in \mathcal{P} : \epsilon\sigma \models \varphi$ iff $\mathcal{L}(\mathcal{P}) \models \varphi$. Thus, any runtime verification approach following this idea implicitly answers the model checking question even before monitoring. Then runtime verification is at least as expensive as model checking.

In general, we can follow a similar idea by having control over the overall complexity using abstractions of the underlying program. An *over-abstraction* or and *over-approximation* of a program $\mathcal{P}$ is a program $\hat{\mathcal{P}}$ such that $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\hat{\mathcal{P}}) \subseteq \Sigma^\omega$.

Definition 1 (Predictive semantics of LTL). *Let $\mathcal{P}$ be a program and let $\hat{\mathcal{P}}$ be an over-approximation of $\mathcal{P}$. Let $u \in \Sigma^*$ denote a finite trace. The* truth value *of u and an LTL_3 formula φ wrt. $\hat{\mathcal{P}}$, denoted by $[u \models_{\hat{\mathcal{P}}} \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:*

$$[u \models_{\hat{\mathcal{P}}} \varphi] = \begin{cases} \top & \textit{if } \forall \sigma \in \Sigma^\omega \textit{ with } u\sigma \in \hat{\mathcal{P}} : u\sigma \models \varphi \\ \bot & \textit{if } \forall \sigma \in \Sigma^\omega \textit{ with } u\sigma \in \hat{\mathcal{P}} : u\sigma \not\models \varphi \\ ? & \textit{else} \end{cases}$$

We write $\text{LTL}_{\mathcal{P}}$ whenever we consider LTL formulas with a predictive semantics.

Remark 1. Let $\hat{\mathcal{P}}$ be an over-approximation of a program $\mathcal{P}$ over Σ, $u \in \Sigma^*$, and $\varphi \in \text{LTL}$.

- Model checking is more precise than RV with the predictive semantics:

$$\mathcal{P} \models \varphi \text{ implies } [u \models_{\hat{\mathcal{P}}} \varphi] \in \{\top, ?\}$$

- RV has no false negatives: $[u \models_{\hat{\mathcal{P}}} \varphi] = \bot$ implies $\mathcal{P} \not\models \varphi$
- The predictive semantics of an LTL formula is more precise than LTL_3:

$$\begin{aligned} [u \models \varphi] = \top &\text{ implies } [u \models_{\hat{\mathcal{P}}} \varphi] = \top \\ [u \models \varphi] = \bot &\text{ implies } [u \models_{\hat{\mathcal{P}}} \varphi] = \bot \end{aligned}$$

The reverse directions are in general not true. Thus, it is possible that a property is violated in the model checking sense but not spotted by RV with predictive semantics.

4 A monitor procedure for LTL$_{\mathcal{P}}$

Now, we develop an automata-based monitor procedure for LTL$_{\mathcal{P}}$. More specifically, for a given over-approximation $\hat{\mathcal{P}}$ of a program $\mathcal{P}$ and formula $\varphi \in$ LTL, we construct a finite Moore machine, $\mathcal{B}^{\varphi}_{\hat{\mathcal{P}}}$ that reads finite traces $u \in \Sigma^*$ and outputs $[u \models_{\hat{\mathcal{P}}} \varphi] \in \mathbb{B}_3$.

For an NBA $\mathcal{A}$, we denote by $\mathcal{A}(q)$ the NBA that coincides with $\mathcal{A}$ except for Q_0, which is defined as $Q_0 = \{q\}$. Fix $\varphi \in$ LTL for the rest of this section and let $\mathcal{A}^{\varphi}$ denote the NBA, which accepts all models of φ, and let $\mathcal{A}^{\neg\varphi}$ denote the NBA, which accepts all counter examples of φ. The corresponding construction is standard [6].

Moreover, fix an over-approximation of a program $\hat{\mathcal{P}}$ for the remainder of this section and let $\mathcal{B}^{\varphi}$ and $\mathcal{B}^{\neg\varphi}$ be the product of the over-approximation with $\mathcal{A}^{\varphi}$ and $\mathcal{A}^{\neg\varphi}$, respectively, i.e., $\mathcal{B}^{\varphi} = \hat{\mathcal{P}} \times \mathcal{A}^{\varphi}$ and $\mathcal{B}^{\neg\varphi} = \hat{\mathcal{P}} \times \mathcal{A}^{\neg\varphi}$. For these automata, we easily observe that for for $u \in \Sigma^*$ and $\delta(Q_0^{\varphi}, u) = \{q_1, \ldots, q_l\}$, we have $[u \models_{\hat{\mathcal{P}}} \varphi] \neq \bot$ iff $\exists q \in \{q_1, \ldots, q_l\}$ *such that* $\mathcal{L}(\mathcal{B}^{\varphi}(q)) \neq \emptyset$. Likewise, we have for the NBA $\mathcal{B}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, F^{\neg\varphi})$ as defined above, for $u \in \Sigma^*$, and $\delta(Q_0^{\neg\varphi}, u) = \{q_1, \ldots, q_l\}$ that $[u \models_{\hat{\mathcal{P}}} \varphi] \neq \top$ iff $\exists q \in \{q_1, \ldots, q_l\}$ *such that* $\mathcal{L}(\mathcal{B}^{\neg\varphi}(q)) \neq \emptyset$.

Following [5], for $\mathcal{B}^{\varphi}$ and $\mathcal{B}^{\neg\varphi}$, we now define a function $\mathcal{F}^{\varphi} : Q^{\varphi} \to \mathbb{B}$ respectively $\mathcal{F}^{\neg\varphi} : Q^{\neg\varphi} \to \mathbb{B}$ (where $\mathbb{B} = \{\top, \bot\}$), assigning to each state q whether the language of the respective automaton starting in state q is not empty. Thus, if $\mathcal{F}^{\varphi}(q) = \top$ holds, then the automaton $\mathcal{B}^{\varphi}$ starting at state q accepts a non-empty language and each finite prefix u leading to state q can be expanded by a run of the over-approximation to satisfy φ.

Using $\mathcal{F}^{\varphi}$ and $\mathcal{F}^{\neg\varphi}$, we define two NFAs $\hat{\mathcal{B}}^{\varphi} = (\Sigma, Q^{\varphi}, Q_0^{\varphi}, \delta^{\varphi}, \hat{F}^{\varphi})$ and $\hat{\mathcal{B}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, \hat{F}^{\neg\varphi})$ where $\hat{F}^{\varphi} = \{q \in Q^{\varphi} \mid \mathcal{F}^{\varphi}(q) = \top\}$ and $\hat{F}^{\neg\varphi} = \{q \in Q^{\neg\varphi} \mid \mathcal{F}^{\neg\varphi}(q) = \top\}$. Then, we have for all $u \in \Sigma^*$:

$$u \in \mathcal{L}(\hat{\mathcal{B}}^{\varphi}) \text{ iff } [u \models_{\hat{\mathcal{P}}} \varphi] \neq \bot \quad \text{and} \quad u \in \mathcal{L}(\hat{\mathcal{B}}^{\neg\varphi}) \text{ iff } [u \models_{\hat{\mathcal{P}}} \varphi] \neq \top$$

Therefore, we can evaluate $[u \models_{\hat{\mathcal{P}}} \varphi]$ as follows: We have $[u \models_{\hat{\mathcal{P}}} \varphi] = \top$ if $u \notin \mathcal{L}(\hat{\mathcal{B}}^{\neg\varphi})$, $[u \models_{\hat{\mathcal{P}}} \varphi] = \bot$ if $u \notin \mathcal{L}(\hat{\mathcal{B}}^{\varphi})$, and $[u \models_{\hat{\mathcal{P}}} \varphi] =?$ if $u \in \mathcal{L}(\hat{\mathcal{B}}^{\varphi})$ and $u \in \mathcal{L}(\hat{\mathcal{B}}^{\neg\varphi})$.

As a final step, we now define a (deterministic) FSM $\mathcal{B}^{\varphi}$ that outputs for each finite string u and formula φ its associated predictive semantics wrt. the over-approximation $\hat{\mathcal{P}}$. Let $\tilde{\mathcal{B}}^{\varphi}$ and $\tilde{\mathcal{B}}^{\neg\varphi}$ be the deterministic versions of $\hat{\mathcal{B}}^{\varphi}$ and $\hat{\mathcal{B}}^{\neg\varphi}$, which can be computed in the standard manner by power-set construction. Now, we define the FSM in question as a product of $\tilde{\mathcal{B}}^{\varphi}$ and $\tilde{\mathcal{B}}^{\neg\varphi}$:

Definition 2 (Predictive Monitor $\mathcal{B}^{\varphi}$ for LTL-formula φ). *Let $\hat{\mathcal{P}}$ be an over-approximation of a program $\mathcal{P}$. Let $\tilde{\mathcal{B}}^{\varphi} = (\Sigma, Q^{\varphi}, \{q_0^{\varphi}\}, \delta^{\varphi}, \tilde{F}^{\varphi})$ and $\tilde{\mathcal{B}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, \{q_0^{\neg\varphi}\}, \delta^{\neg\varphi}, \tilde{F}^{\neg\varphi})$ be the DFAs which correspond to the two NFAs $\hat{\mathcal{B}}^{\varphi}$ and $\hat{\mathcal{B}}^{\neg\varphi}$ as defined before. Then we define the* predictive monitor $\mathcal{B}^{\varphi} = \tilde{\mathcal{B}}^{\varphi} \times \tilde{\mathcal{B}}^{\neg\varphi}$ *for φ with respect to $\hat{\mathcal{P}}$ as the minimized version of the FSM $(\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$,*

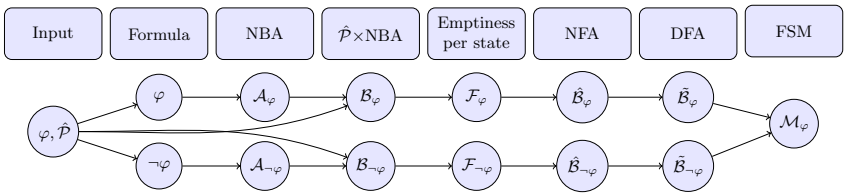

Fig. 1. The procedure for getting $[u \models_{\hat{\mathcal{P}}} \varphi]$ for a given φ and over-approximation $\hat{\mathcal{P}}$

where $\bar{Q} = Q^\varphi \times Q^{\neg\varphi}$, $\bar{q}_0 = (q_0^\varphi, q_0^{\neg\varphi})$, $\bar{\delta}((q,q'),a) = (\delta^\varphi(q,a), \delta^{\neg\varphi}(q',a))$, *and* $\bar{\lambda} : \bar{Q} \to \mathbb{B}_3$ *is defined by*

$$\bar{\lambda}((q,q')) = \begin{cases} \top & \text{if } q' \notin \tilde{F}^{\neg\varphi} \\ \bot & \text{if } q \notin \tilde{F}^{\varphi} \\ ? & \text{if } q \in \tilde{F}^{\varphi} \text{ and } q' \in \tilde{F}^{\neg\varphi}. \end{cases}$$

We sum up our entire construction in Fig. 1 and conclude with the following correctness theorem.

Theorem 1. *Let* $\hat{\mathcal{P}}$ *be an over-approximation of a program* $\mathcal{P}$, $\varphi \in$ *LTL, and let* $\mathcal{B}^\varphi = (\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$ *be the corresponding monitor. Then, for all* $u \in \Sigma^*$*:* $[u \models_{\hat{\mathcal{P}}} \varphi] = \bar{\lambda}(\bar{\delta}(\bar{q}_0, u))$.

Complexity. Consider Fig. 1: Given φ, step 1 requires us to replicate φ and to negate it, i.e., it is linear in the original size. Step 2, the construction of the NBAs, causes an exponential blow-up in the worst-case. Step 3 multiplies the size of the automaton with the size of the over-approximation $\hat{\mathcal{P}}$. Steps 4 and 5, leading to $\hat{\mathcal{B}}^\varphi$ and $\hat{\mathcal{B}}^{\neg\varphi}$, do not change the size of the original automata. Then, computing the deterministic automata of step 6, might again require an exponential blow-up in size. In total the FSM of step 7 will have double exponential size with respect to $|\varphi|$ and single exponential size with respect to $\hat{\mathcal{P}}$. Note that steps 6 and 7 can easily be done on-the-fly.

References

1. Kamp, H.W.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles (1968)
2. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, New York (1995)
3. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: CAV03. Volume 2725 of LNCS., Boulder, CO, USA, Springer (July 2003) 27–39
4. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation **20**(3) (2010) 651–674
5. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM TOSEM **20**(4) (jul 2011)
6. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Symposium on Logic in Computer Science (LICS'86), Washington, D.C., USA, IEEE Computer Society Press (June 1986) 332–345