Monitoring of real-time properties

Andreas Bauer, Martin Leucker, and Christian Schallhart

Institut für Informatik, Technische Universität München

Abstract. This paper presents a construction for runtime monitors that check real-time properties expressed in TLTL (i. e., timed LTL). Due to D'Souza's results, TLTL can be considered a natural extension of LTL towards real-time. Moreover, a typical obstacle in runtime verification is solved both for untimed and timed formulae, in that standard models of linear temporal logic are infinite traces, whereas in runtime verification only finite system behaviours are at hand. Therefore, a 3-valued semantics (*true, false, inconclusive*) for LTL and TLTL on finite traces is defined that resembles the infinite trace semantics in a suitable and intuitive manner. Then, the paper describes how to construct, given a (T)LTL formula, an optimal deterministic monitor with three output symbols that reads a finite trace and yields its according 3-valued (T)LTL semantics. Notably, the monitor rejects a trace as early as possible, in that any minimal bad prefix results in *false* as a return value.

1 Introduction

Runtime verification [10] is becoming a popular tool to complement verification techniques such as model checking and testing, especially for so-called black box systems. In a nutshell, runtime verification works as follows. A correctness property φ, usually formulated in some linear temporal logic, such as LTL [21], is given and a so-called *monitor* that accepts all models for φ is automatically generated. The system under scrutiny as well as the generated monitor are then executed in parallel, such that the monitor observes the system's behaviour. System behaviour which violates property φ is then detected by the monitor and an according alarm signal is returned.

Monitors can be employed in different phases of system development: In the testing phase [8], the system is executed with typical inputs and monitors are observed for complaints. At customer's site, monitors check for bugs that escaped the testing phase and may trigger recovery actions [5].

Various runtime verification approaches for LTL have been proposed already [14, 17, 18, 16, 24]. However, the current approaches suffer—to our opinion—from the treatment of the following obstacle: Notably, the semantics of LTL is defined over infinite (behavioural) traces whereas monitoring a running system allows an at most finite view. In consequence, various authors have proposed custom interpretations of LTL over finite traces using *weak* and *strong semantics*: the weak interpretation of a formula φ w. r. t. a finite trace u is that if up to the point where u ends, "nothing has yet gone wrong", φ holds. In the strong view, φ holds only if it evaluates to $true$ within u. Eisner et al. give a good overview on the topic [13]. However, good examples can be found for each of the interpretations and—at the same time—also examples that the chosen approach is misleading.

In this paper, we propose a simple, yet—as we find—convincing way to overcome this obstacle. Instead of trying to define a two-valued semantics for LTL on finite traces, we define a three valued semantics, using values $true$, $false$, and ?, where the latter denotes *inconclusive*. Given a finite string u and a formula φ, the truth values are defined as expected: if there is no continuation of u satisfying φ, the value is false. If every continuation of u satisfies φ, we go for true. Otherwise, we say ?, since the observations so far are just inconclusive to say either true or false.

We argue that it is important to work with three instead of two truth values: Consider, for instance, the property $G\neg p$ stating that no state satisfying p should occur. Clearly, when p is observed, the monitor should complain. As long as p does not hold, it is misleading to say that the formula is true, since the next observation might already violate the formula. On the other hand, consider the formula $\neg p\, U\, \mathit{init}$ stating that nothing bad (p) should happen before the init function is called. If, indeed, the init function has been called and no p has been observed before, the formula is true, regardless what will happen in the future. For testing and verification, it is important to know whether some property is indeed true or whether the current observation is just inconclusive.

Thus, in this paper, we propose a 3-valued logic, LTL_3, which can be interpreted over finite traces based on the standard semantics of LTL for infinite traces. Furthermore, we describe how to construct, given an LTL formula, a (deterministic) finite state machine (FSM) with three output symbols. This automaton reads finite traces and yields their 3-valued LTL semantics. Hence, it can be directly deployed for runtime verification. Standard minimisation techniques for FSMs can be used to obtain an *optimal* FSM w. r. t. number of states.

The nature of our 3-valued semantics for LTL rounds off the study of safety properties in terms of automata carried out in [19] from a temporal logic perspective. In [19] , a *bad prefix* (of a Büchi automaton), is defined as a finite prefix which cannot be the prefix of any accepting trace. Dually, a *good prefix* is a finite prefix such that any infinite extension of the trace will be accepted. It is exactly this classification that forms the basis of our 3-valued semantics: "bad prefixes" (of formulas) are mapped to false, "good prefixes" evaluate to true, while the remaining prefixes yield ?. Thus, monitors for 3-valued formulas classify prefixes as one of $\mathit{good} = \mathit{true}$, $\mathit{bad} = \mathit{false}$, or ? (neither good nor bad).

Since an extension of a bad (good) prefix is bad (good, respectively), there is a *minimal* bad (good) prefix for every bad (good) prefix. In runtime verification, one is interested in getting information already for minimal prefixes and one solution was worked out in [11]. However, all "bad prefixes" for a formula φ gives rise to false–also minimal ones. Thus, the correctness of our monitor procedures for LTL and TLTL ensures that already for minimal good or bad prefixes one of true or false is obtained. Altogether, we get a coherent study of (not only safety) LTL properties based on finite prefixes together with *optimal* acceptors, as they are called in [11], based on elementary results for LTL and automata theory.

To make our result easily accessible to the reader and to complete the picture started in [11], our concepts are first developed in the setting of LTL. However, the main concern of this paper are real-time systems. Therefore, we develop our ideas also for TLTL, a logic introduced in [22], which, as argued by D'Souza in [12] can be considered a natural counterpart of LTL in the timed setting. Hence, for a TLTL formula a monitor is constructed which operates over finite *timed* traces. Again, by correctness of our construction, monitors signal faults or satisfaction "as early as possible".

While the general scheme, as we show, is also applicable in the timed setting, the monitor construction is technically much more involved. Automata for TLTL employ so-called *event recording* and *event predicting* clocks. Since in runtime verification the future of a trace is not known, predicting events are difficult to handle. We introduce *symbolic runs* and show their benefit for checking promises efficiently, avoiding the translation of event-clock automata to (predicting-free) timed automata, which are harder to employ in runtime verification.

[15] studies monitor generation based on LTL enriched with a freeze quantifier for time. In [25, 7], *fault diagnosis* for timed systems is examined, a problem that is more complicated than runtime verification. However, only timed automata or event recording automata are used, no prediction of events is supported. TLTL is event-based, meaning that the system emits events when the system's state has changed. In [20] monitoring of continuous signals is considered, which is intrinsicly different to observing discrete signals in a continuous time domain.

All of the work mentioned so far employs a 2-valued semantics, except for [11], which does not discuss the role of minimal prefixes for runtime verification that our approach offers for free thanks to the 3-valued semantics.

We have implemented the untimed setting and validated our approach examining a real-world case study. The monitor generator as open-source, exemplifying material, a case study, as well as a full version of the paper is available from `http://runtime.in.tum.de/`.

Clearly, our 3-valued semantics for finite prefixes of sets of infinite traces has nothing to do with the work on 3-valued semantics for LTL on a *single* infinite trace, studied for example in [9].

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, i.e., a_i is a possibly empty set of propositions taken from AP. 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 $w = a_0a_1 \ldots \in \Sigma^\omega$ 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, $Q_0 \in Q$ 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 $\delta : Q \times \Sigma \to 2^Q$, as usual, to $\delta' : 2^Q \times \Sigma^* \to 2^Q$ by $\delta'(Q', \epsilon) = Q'$where $Q' \subseteq Q$and $\delta'(Q', ua) = \bigcup_{q' \in \delta'(Q', u)} \delta(q', a)$. To simplify notation, we use δ for both δ and δ'. A NBA is called *deterministic* iff for all $q \in Q$, $a \in \Sigma$, $|\delta(q, a)| = 1$, and $|Q_0| = 1$. We use DBA to denote a deterministic Büchi automaton.

A *run* of an automaton $\mathcal{A}$ on a word $w = a_1 \ldots \in \Sigma^\omega$ is a sequence of states and actions $\rho = q_0a_1q_1 \ldots$, where q_0 is an initial state of $\mathcal{A}$ and for all $i \in \mathbb{N}$ we have

$q_{i+1} \in \delta(q_i, a)$. For a run ρ, let Inf(ρ) denote the states visited infinitely often. A run ρ of a NBA $\mathcal{A}$ is called *accepting* iff Inf$(\rho) \cap F \neq \emptyset$.

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

A NFA is called *deterministic* iff for all $q \in Q$, $a \in \Sigma$, $|\delta(q,a)| = 1$, and $|Q_0| = 1$. We use DFA to denote a deterministic finite automaton.

Finally, let us recall the notion of a *Moore machine*, also called *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, $Q_0 \in Q$, 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, rather than by input symbols. As before, δ extends to the domain of words as expected. For a deterministic Moore machine, 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.

3 Three-valued LTL in the untimed setting

Semantics To overcome difficulties in defining an adequate boolean semantics for LTL on finite traces, we propose a 3-valued semantics. The intuition is as follows: in theory, we observe an infinite sequence w of some system. For a given formula φ, thus either $w \models \varphi$ or not. In practice, however, we can only observe a finite prefix u of w. Consequently, we let the semantics of u and φ be true, if $uw' \models \varphi$ for every possible future extension w'. On the other hand, if uw' is not a model of φ for all possible infinite continuations w' of u, we define the semantics of u and φ as false. In the remaining case, the truth value of uw' and φ depends on w'. Thus, we define the semantics of u with respect to φ to be *inconclusive*, denoted by ?, to signal that u itself is not sufficient to determine how φ will evaluate in any possible future which is prefixed with u.

Formally, we define our 3-valued semantics in terms of LTL$_3$ over the set of truth values $\mathbb{B}_3 = \{\bot, ?, \top\}$ as follows:

Definition 1 (3-valued semantics of LTL). *Let $u \in \Sigma^*$ denote a finite trace. The* truth value *of a LTL$_3$ formula φ w. r. t. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:*

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

A monitor procedure for LTL$_3$ Now, we develop an automata-based monitor procedure for LTL$_3$. More specifically, for a given formula $\varphi \in$ LTL$_3$, we construct a finite Moore machine, $\bar{\mathcal{A}}^\varphi$ that reads finite traces $u \in \Sigma^*$ and outputs $[u \models \varphi]$, thus a value in $\mathbb{B}_3$.

For a 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 and explained, for example in [27]. For these automata, we observe:

Lemma 1. *Let $\mathcal{A}^\varphi = (\Sigma, Q^\varphi, Q_0^\varphi, \delta^\varphi, F^\varphi)$ denote the NBA such that $\mathcal{L}(\mathcal{A}^\varphi) = \mathcal{L}(\varphi)$. For $u \in \Sigma^*$, let $\delta(Q_0^\varphi, u) = \{q_1, \dots, q_l\}$. Then*

$$[u \models \varphi] \neq \bot \text{ iff } \exists q \in \{q_1, \dots, q_l\} \text{ such that } \mathcal{L}(\mathcal{A}^\varphi(q)) \neq \emptyset.$$

Lemma 2. *Let $\mathcal{A}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, F^{\neg\varphi})$ denote the NBA such that $\mathcal{L}(\mathcal{A}^{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. For $u \in \Sigma^*$, let $\delta(Q_0^{\neg\varphi}, u) = \{q_1, \dots, q_l\}$. Then*

$$[u \models \varphi] \neq \top \text{ iff } \exists q \in \{q_1, \dots, q_l\} \text{ such that } \mathcal{L}(\mathcal{A}^{\neg\varphi}(q)) \neq \emptyset.$$

Correctness of the first lemma follows directly from the definition of acceptance for Büchi automata and the second lemma rephrases the first one by substituting $\neg\varphi$ for φ.

For $\mathcal{A}^\varphi$ and $\mathcal{A}^{\neg\varphi}$, 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. Using $\mathcal{F}^\varphi$ and $\mathcal{F}^{\neg\varphi}$, we define two NFAs $\hat{\mathcal{A}}^\varphi = (\Sigma, Q^\varphi, Q_0^\varphi, \delta^\varphi, \hat{F}^\varphi)$ and $\hat{\mathcal{A}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, \hat{F}^{\neg\varphi})$ where $\hat{F}^\varphi = \{q \in Q^\varphi \mid \mathcal{F}^\varphi(q) = \top\}$ and $\hat{F}^{\neg\varphi} = \{q \in Q^{\neg\varphi} \mid \mathcal{F}^{\neg\varphi}(q) = \top\}$.

$\hat{\mathcal{A}}^\varphi$, resp. $\hat{\mathcal{A}}^{\neg\varphi}$, accept the finite traces u for which $[u \models \varphi]$ evaluates to $\neq \bot$ and, respectively, $\neq \top$.

Lemma 3. *Using the notation as before, we have for all $u \in \Sigma^*$:*

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

Therefore, we can evaluate $[u \models \varphi]$ according to Lemma 3 as follows.

Lemma 4. *With the notation as before, we have $[u \models \varphi] = \top$ if $u \notin \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$, $[u \models \varphi] = \bot$ if $u \notin \mathcal{L}(\hat{\mathcal{A}}^\varphi)$, and $[u \models \varphi] =?$ if $u \in \mathcal{L}(\hat{\mathcal{A}}^\varphi)$ and $u \in \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$.*

The lemma yields a simple procedure to evaluate the semantics of φ for a given finite trace u: we evaluate both $u \in \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$ and $u \in \mathcal{L}(\hat{\mathcal{A}}^\varphi)$ and use Lemma 4 to determine $[u \models \varphi]$. As a final step, we now define a (deterministic) FSM $\bar{\mathcal{A}}^\varphi$ that outputs for each finite string u its associated 3-valued semantical evaluation with respect to some LTL-formula φ.

Let $\tilde{\mathcal{A}}^\varphi$ and $\tilde{\mathcal{A}}^{\neg\varphi}$ be the deterministic versions of $\hat{\mathcal{A}}^\varphi$ and $\hat{\mathcal{A}}^{\neg\varphi}$, which can be computed in the standard manner by power-set construction. Now, we define the FSM in question as a product of $\tilde{\mathcal{A}}^\varphi$ and $\tilde{\mathcal{A}}^{\neg\varphi}$:

Definition 2 (Monitor $\bar{\mathcal{A}}^\varphi$ for a LTL-formula φ). *Let $\tilde{\mathcal{A}}^\varphi = (\Sigma, Q^\varphi, \{q_0^\varphi\}, \delta^\varphi, \hat{F}^\varphi)$ and $\tilde{\mathcal{A}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, \{q_0^{\neg\varphi}\}, \delta^{\neg\varphi}, \hat{F}^{\neg\varphi})$ be the DFAs which correspond to the two NFAs $\hat{\mathcal{A}}^\varphi$ and $\hat{\mathcal{A}}^{\neg\varphi}$ as defined for Lemma 3. Then we define the* monitor *$\bar{\mathcal{A}}^\varphi = \tilde{\mathcal{A}}^\varphi \times \tilde{\mathcal{A}}^{\neg\varphi}$ as the FSM $(\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$, 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 by formulating the correctness theorem.

Theorem 1. *Let $\varphi \in LTL_3$ and let $\bar{\mathcal{A}}^\varphi = (\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$ be the corresponding monitor. Then, for all $u \in \Sigma^*$ the following holds: $[u \models \varphi] = \bar{\lambda}(\bar{\delta}(\bar{q}_0, u))$.*

$$\begin{tikzcd}
\text{Input} & \text{(1) Formula} & \text{(2) NBA} & \text{(3) Emptiness per state} & \text{(4) NFA} & \text{(5) DFA} & \text{(6) FSM} \\
 & \varphi \arrow[r] & \mathcal{A}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \hat{\mathcal{A}}^{\varphi} \arrow[r] & \tilde{\mathcal{A}}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & \bar{\mathcal{A}} \\
 & \neg\varphi \arrow[r] & \mathcal{A}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \hat{\mathcal{A}}^{\neg\varphi} \arrow[r] & \tilde{\mathcal{A}}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Fig. 1. The procedure for getting $[u \models \varphi]$ for a given φ

Complexity. Let us study the size of the resulting FSM. 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. Steps 3 and 4, leading to $\hat{\mathcal{A}}^{\varphi}$ and $\hat{\mathcal{A}}^{\neg\varphi}$, do not change the size of the original automata. Then, computing the deterministic automata of step 5, might again require an exponential blow-up in size. In total the FSM of step 6 will have double exponential size with respect to $|\varphi|$.

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 w. r. t. 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.

That said, we have implemented the determinisation of NFAs and the product for obtaining $\bar{\mathcal{A}}$ (steps 4–6) in an *on-the-fly* fashion. This technique is well known for example in compiler construction [1]. Our examples confirm huge savings in memory consumption.

4 Three-valued LTL in the timed setting—TLTL

In this part, we extend the approach developed in the preceding section to the timed setting. Thus, the goal is to dynamically check real-time specifications formulated in a timed temporal logic. We use timed LTL (TLTL for short), a logic introduced in [22], in the form presented in [23]. The language expressible by a TLTL formula can be defined by *event-clock automata* [4], a subclass of *timed automata*. It was shown in [12] that TLTL corresponds exactly to the class of languages definable in first-order logic interpreted over timed words. Thus, it can be considered as the natural counterpart of LTL for the timed setting. Given the translation to event-clock automata in the literature [23], it is promising to base our timed runtime verification approach on TLTL and event-clock automata.

4.1 Preliminaries

Let us fix an alphabet Σ of actions for the rest of this section. In the timed setting, every symbol $a \in \Sigma$ is associated with an *event-recording clock*, x_a, and an *event-predicting clock*, y_a. An (infinite) *timed word* w over the alphabet Σ is an (infinite) sequence of *timed events* $(a_0, t_0)(a_1, t_1) \ldots$ consisting of symbols $a_i \in \Sigma$, and non-negative numbers $t_i \in \mathbb{R}^{\geq 0}$, such that for each $i \in \mathbb{N}$, $t_i < t_{i+1}$ (*strict monotonicity*), and for all $t \in \mathbb{R}^{\geq 0}$ there is an $i \in \mathbb{N}$ such that $t_i > t$ (*progress*). Furthermore, for w as above, we call its sequence of actions (the projection to the first component) the *untimed word* of w, denoted by $ut(w)$.

To simplify notation, we abbreviate $(\Sigma \times \mathbb{R}^{\geq 0})$ by $T\Sigma$. Thus, a finite timed word is an element of $T\Sigma^*$ and the domain of infinite timed words is denoted by $T\Sigma^{\omega}$. Given an (infinite) timed word w, the value of the event-recording clock variable x_a at position j of w equals $t_j - t_i$, where i represents the last position preceding j such that $a_i = a$. If no such position exists, then the value of x_a remains undefined, denoted by $\bot$. The event-predicting clock variable y_a equals $t_j - t_i$, where j represents the next position after i such that $a_j = a$. If no such position exists, again, the variable remains undefined. The set of all event-clocks is denoted by $C_\Sigma = \{x_a, y_a \mid a \in \Sigma\}$. A *clock valuation function* over a timed word w, $\gamma_i : C_\Sigma \to \mathbb{R}^{\geq 0} \cup \{\bot\}$ assigns a positive real, or undefined value to each clock variable corresponding to position i. We abbreviate $\mathbb{R}^{\geq 0} \cup \{\bot\}$ by $T_\bot$.

A *clock constraint* compares a clock value to a natural number. Let $\Psi(C_\Sigma)$ denote the set of clock constraints over C_Σ. Formally, a clock constraint $\psi \in \Psi(C_\Sigma)$ is a conjunction of atomic formulae of the form $z \bowtie c$, where $z \in C_\Sigma$, $\bowtie \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N}$. Given a clock constraint ψ and a clock valuation function γ, we write $\gamma \models \psi$ to denote that according to γ, constraint ψ is fulfilled, where $\bot \bowtie c$ for $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$ does not hold, and the remaining cases are defined in the expected manner.

4.2 Syntax and semantics of TLTL$_3$

Let Σ be a finite set of actions. A set of formulas φ of TLTL is defined by the grammar

$$\varphi ::= true \mid a \mid \lhd_a \in I \mid \rhd_a \in I \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\, U\, \varphi \mid X\varphi \quad (a \in \Sigma),$$

where $\lhd_a$ is the operator that measures the time elapsed since the last occurrence of a, and $\rhd_a$ the operator that predicts the next occurrence of a within a timed interval $I \in \mathcal{I}$. The set of intervals $\mathcal{I}$ contains intervals of the form (l, r), $[l, r)$, $(l, r]$, or $[l, r]$, where $l, r \in \mathbb{R}^{\geq 0} \cup \{\infty\}$. Without loss of generality, we assume $l < r$, except for $[l, r]$, and for intervals $(l, r]$, or $[l, r]$ that $r \neq \infty$. To simplify notation, we use $[\![$ and $]\!]$ for interval borders which can either be (or [, respectively),].

The semantics of TLTL formulae are defined inductively over infinite timed words $w \in T\Sigma^\omega$, where $w = (a_0, t_0)(a_1, t_1)$, and $i \in \mathbb{N}^{\geq 0}$ as follows: $w, i \models true$, $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$, $w, i \models a$ iff $a(i) = a$, $w, i \models \lhd_a \in I$ iff $\gamma_i(x_a) \in I$, $w, i \models \rhd_a \in I$ iff $\gamma_i(y_a) \in I$, $w, i \models \varphi_1 \vee \varphi_2$ iff $w, i \models \varphi_1$ or $w, i \models \varphi_2$, $w, i \models \varphi_1 U \varphi_2$ iff $\exists k \geq i$ with $w, k \models \varphi_2$ and $\forall l : (i \leq l < k \wedge w, l \models \varphi_1$, $w, i \models X\varphi$ iff $w, i + 1 \models \varphi$. Further, let $w \models \varphi$, iff $w, 0 \models \varphi$.

Analogously to the untimed case, we now define a 3-valued semantics for TLTL, from this point onwards denoted as TLTL$_3$, as follows:

Definition 3. *Let $u \in T\Sigma^*$ denote a finite timed trace. The truth value of a TLTL$_3$ formula φ w. r. t. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:*

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

4.3 Symbolic runs of event-clock automata

We first recall the definition of an event-clock automaton: Given a finite set of clocks, C_Σ, we define an event-clock automaton as a finite state automaton whose edges are annotated both with input symbols and with clock constraints as $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$, where Σ is a finite input alphabet, Q a finite set of states, $Q_0 \subseteq Q$ are initial states, $F \subseteq 2^Q$ is a set of accepting states (generalised Büchi acceptance condition), and $E \subseteq Q \times \Sigma \times \Psi(C_\Sigma) \times Q$

a set of transitions. An edge $e = (q, a, \psi, q')$ represents a transition from source state q upon symbol a to destination q', where the clock constraint ψ then specifies when this transition is enabled. For an event-clock automaton $\mathcal{A}$, let $K_{\mathcal{A}}$ denote the biggest constant appearing in some constraint of $\mathcal{A}$; we write K when $\mathcal{A}$ is clear from the context.

A *timed run* θ of an automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ over a timed word $w \in T\Sigma^\omega$ starting in (q_0, γ_0) is an infinite sequence of state-valuation tuples and transitions as follows: $(q_0, \gamma_0) \overset{\alpha_1}{\to} (q_1, \gamma_1) \overset{\alpha_2}{\to} \ldots$ with $q_i \in Q$, and γ_i being the evaluation function assigning for every element from Σ the value of the recording and predicting event clocks corresponding to α_i, where $\alpha_i \in T\Sigma$ is a timed event of the form $(a_i \in \Sigma, t_i \in \mathbb{R}^{\geq 0})$, and for all $i \geq 1$ there is a transition in E of the form $(q_{i-1}, a_i, \psi, q_i)$ such that $\gamma_i \models \psi$. $\mathcal{A}_{ec}$ accepts θ, iff for each $F_i \in F$, a state $q \in F_i$ exists such that q occurs infinitely often in θ.

γ_0 is *initial* (w.r.t. w) if $\gamma_0(x_a) = \bot$ and $\gamma_0(y_a) = i$ if $\alpha_i = (a, t_i)$ and for $j < i$ and $\alpha_j = (a_j, t_j)$, $a_j \neq a$, and $\gamma_0(y_a) = \bot$ if a does not occur in w. Then, the timed language accepted by $\mathcal{A}_{ec}$, denoted as $\mathcal{L}(\mathcal{A}_{ec})$, is the set of timed words for which an accepting run of $\mathcal{A}_{ec}$ exists starting in (q_0, γ_0), for some $q_0 \in Q_0$ and the initial γ_0.

For runtime verification predicting clock variables pose a problem, since information about the future occurrence of an action a is predicted, but this information is not available yet. We solve this problem by representing the value of some predicting clock variable *symbolically*.

A *symbolic clock valuation function* $\Gamma : C_\Sigma \to T_\bot \cup \mathcal{I}$ assigns a positive real, or undefined value to each recording clock variable and an *interval* or undefined value to each predicting clock variable. The interval constrains the possible values of a predicting variable. To simplify notation, we identify $\Gamma(y_a) = (l, r)$ with the constraint $y_a > l \wedge y_a < r$ (and similarly for borders [and]).

For a symbolic clock evaluation Γ, we define the following three operations: time *elapse*, *reset*, and *conjunction*. Given an *elapsed* time $t \in \mathbb{R}^{\geq 0}$, $\Gamma' = \Gamma + t$, where $\Gamma'(x_a) = \Gamma(x_a) + t$ and for $\Gamma(y_a) = [\![l, r]\!]$, we set $\Gamma'(y_a) = [\![l \dot{-} t, r - t]\!]$, where $\dot{-}$ yields at least 0. If $r - t < 0$, then Γ' is invalid. Γ *reset* by action a, denoted by $\Gamma{\downarrow}a$, sets $x_a = 0$ and removes all constraints on y_a, and we set $\Gamma'(y_a) = [0, \infty)$ and $\Gamma'(z_b) = \Gamma(z_b)$ for all $b \neq a$. The *conjunction* of Γ with constraint ψ yields $\Gamma' = \Gamma \wedge \psi$, where each predicting clock y_a is combined with the constraints of ψ which involve y_a, i. e., for $a \in \Sigma$, $\Gamma'(y_a) = \Gamma(y_a) \wedge \bigwedge\{y_a \bowtie c \subseteq \psi\}$. We call Γ' invalid, if for some y_a, $\Gamma'(y_a)$ is not satisfiable.

Furthermore, a transition $(q, a, \psi, q') \in E$ is *applicable* to a pair (q, Γ), if the constraints $x_b \bowtie c$ in ψ are satisfied by Γ, for all $b \in \Sigma$, and $0 \in \Gamma(y_a)$. If $(q, a, \psi, q') \in E$ is applicable, then the corresponding successor of (q, Γ) is (q', Γ'), where $\Gamma' = (\Gamma{\downarrow}a) \wedge \psi$.

A *symbolic timed run* Θ of an automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ over a timed word $w \in T\Sigma^\omega$ starting in (q_0, Γ_0) is an infinite sequence of state-symbolic-valuation tuples and transitions as follows: $(q_0, \Gamma_0) \overset{\alpha_1}{\to} (q_1, \Gamma_1) \overset{\alpha_2}{\to} \ldots$ with $q_i \in Q$, and Γ_i being a symbolic valuation function, where for each $(q_{i-1}, \Gamma_{i-1}) \overset{(a_i, t_i)}{\to} (q_i, \Gamma_i)$, there exists some transition $(q_{i-1}, a_i, \psi, q_i)$ applicable to $(q_{i-1}, \Gamma_{i-1} + t_i)$ and (q_i, Γ_i) is the result of this application. The notion of acceptance for symbolic runs corresponds to that of runs, i. e., for each $F_i \in F$ there is some $q \in F_i$ occurring infinitely often.

We call Γ_0 *initial* if for $a \in \Sigma$, $\Gamma_0(x_a) = \bot$ and $\Gamma_0(y_a) = [0, \infty)$.

Theorem 2. *Let $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ be an event-clock automaton and $w \in T\Sigma^\omega$. Then, there is an accepting run on w starting in (q_0, γ_0) iff there is a symbolic accepting run on w starting in (q_0, Γ_0).*

The important fact about the previous theorem is that γ_0 is dependent on w (since each predicting clock y_a has to be initialised to match the first occurrence of a), while Γ_0 is independent of w. Thus, symbolic runs are a suitable device for runtime verification.

4.4 A monitor procedure for TLTL$_3$

We can assume that for some property φ as well as its negation, an event-clock automaton is given, accepting precisely the models respectively counterexamples of φ respectively $\neg\varphi$ (see [23] for details). Looking at the scheme developed in the untimed setting, we are now tempted to check for every state q of the event-clock automaton, whether the language accepted from state q is empty. However, this would yield wrong conclusions, as can be seen in Fig. 2. While the language accepted in state 2 is non-empty and, despite, state 2 is reachable, the automaton does not accept any word when starting in state 0. The constraint when passing from 1 to 2 requires the clock x_a to be at least 2. This, however, restricts the loop in state 2 to be taken.

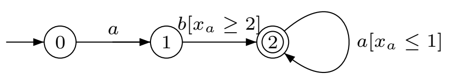
$$\begin{tikzcd} \arrow[r] & 0 \arrow[r, "a"] & 1 \arrow[r, "{b[x_a \geq 2]}"] & \boxed{2} \arrow[loop right, "{a[x_a \leq 1]}"] \end{tikzcd}$$

Fig. 2. Event-clock automaton

We therefore decided to work on the so-called region automaton (for alternatives see Remark 2 on page 11). Recall that K denotes the biggest constant occurring in some constraint of the event-clock automaton. Two clock valuations γ_1, γ_2 are in the same region, denoted by $\gamma_1 \equiv \gamma_2$ iff

- for all $z \in C_\Sigma$, $\gamma_1(z) = \bot$ iff $\gamma_2(z) = \bot$, and (*agreement on undefined*)
- for all $z \in C_\Sigma$, if $\gamma_1(z) \leq K$ or $\gamma_2(z) \leq K$, then $\lfloor \gamma_1(z) \rfloor = \lfloor \gamma_2(z) \rfloor$, and (*agreement on integral part*)
- for all $a \in \Sigma$, let $\langle\langle \gamma(x_a) \rangle\rangle = \lceil x_a \rceil - \gamma(x_a)$ and $\langle\langle \gamma(y_a) \rangle\rangle = \gamma(y_a) - \lfloor y_a \rfloor$. Then, for all $z_1, z_2 \in C_\Sigma$ with $\gamma_1(z_1) \leq K$ and $\gamma_2(z_2) \leq K$,
  - $\langle\langle \gamma_1(z_1) \rangle\rangle = 0$ iff $\langle\langle \gamma_2(z_1) \rangle\rangle = 0$
  - $\langle\langle \gamma_1(z_1) \rangle\rangle \leq \langle\langle \gamma_1(z_2) \rangle\rangle$ iff $\langle\langle \gamma_2(z_1) \rangle\rangle \leq \langle\langle \gamma_2(z_2) \rangle\rangle$. (*agreement on order of fractions*)

A *clock region* is an equivalence class of $\equiv$. Let $\mathcal{R}$ denote the set of all regions.

The key property of the region equivalence is *stability* [3]: given state s and two equivalent valuations γ_1 and γ_2, then (s', γ') is an a-successor of (s, γ_1) iff it is one of (s, γ_2), too. By induction, this can be lifted to infinite runs.This yields:

Lemma 5. *Let $\mathcal{A}_{ec}$ be an event-clock automaton. Let q be some state of $\mathcal{A}_{ec}$ and γ_1, γ_2 two valuations with $\gamma_1 \equiv \gamma_2$. Let $\bar{w} \in \Sigma^\omega$. Then, there exists an accepting run on some infinite timed word $w_1 \in T\Sigma^\omega$ with $ut(w_1) = \bar{w}$ starting in (q, γ_1) iff there exists an accepting run on some infinite timed word $w_2 \in T\Sigma^\omega$ with $ut(w_2) = \bar{w}$ starting in (q, γ_2).*

Note that the so-called *zone equivalence* [2] is not stable. Thus, *zone automata*, while successfully used in model checking tools such as Uppaal [6], do not satisfy our needs.

For completeness, we give the translation of an event-clock automaton to a region automaton, as presented in [23], whose states actually serve their purpose in our approach, because of the previous lemma.

A clock region κ_2 is a *time successor* of a clock region κ_1, denoted by $\kappa_2 \in TS(\kappa_1)$, iff for all $\gamma \in \kappa_1$ there is some $t \in \mathbb{R}^{\geq 0}$ such that $\gamma + t \in \kappa_2$. Here, $\gamma' = \gamma + t$ is defined as $\gamma'(x_a) = \gamma(x_a) + t$ and $\gamma'(y_a) = \gamma(y_a) - t$. To simplify notation, let us fix an event-clock automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$. The region automaton of $\mathcal{A}_{ec}$ is the (generalised) Büchi automaton $R(\mathcal{A}_{ec}) = (\Sigma^r, Q^r, Q_0^r, E^r, F^r)$, where

- $Q^r\{(l, \kappa, \zeta) \mid l \in Q, \kappa \in \mathcal{R}, \zeta \in \{t, d\}\}$ is the set of states
- $Q_0^r = \{(l, \kappa, \zeta) \in Q^r \mid l \in Q_0, \forall a \in \Sigma, \kappa(x_a) = \bot, \zeta = d\}$ is the set of initial states
- $\Sigma^r = \Sigma \cup \{\epsilon\}$
- $E^r = E_d^r \cup E_t^r$ is the union of untimed and timed transitions, where
  - $E_d^r = \{((l_1, \kappa_1, t), (l_2, \kappa_2, d), a) \mid (l_1, a, \psi, l_2) \in E$ and $\exists \kappa_3$ s. t. $\kappa_1 = \kappa_3[y_a := 0], \kappa_2 = \kappa_3[x_a := 0]$, and $\kappa_3 \models \psi\}$
  - $E_t^r = \{((l, \kappa_1, d), (l, \kappa_2, t), \epsilon) \mid \kappa_2 \in TS(\kappa_1)\}$
- $F^r = \{F_i^r \mid F_i \in F\} \cup \{F_{x_a} \mid \lhd_a \in I \in Sub(\varphi)\} \cup \{F_{y_a} \mid \rhd_a \in I \in Sub(\varphi)\}$,
  - where for $F_i \in F$, $F_i^r = \{(l, \kappa, \zeta) \mid l \in F_i\}$
  - $F_{x_a} = \{(l, \kappa, \zeta \mid \forall \gamma \in \kappa\, \gamma(x_a) = 0 \vee \gamma(x_a) > c \vee \gamma(x_a) = \bot\}$
  - $F_{y_a} = \{(l, \kappa, \zeta \mid \forall \gamma \in \kappa\, \gamma(y_a) = 0 \vee \gamma(y_a) = \bot\}$

Note that the region automaton as defined here is a Büchi automaton and thus, the accepted language is a sequence of (untimed) words over Σ. Thus, it is easy to compute for every state, whether the accepted (untimed) language is empty or not. For every state (l, κ, ζ) with a non-empty language, stability now guarantees that for each $\gamma \in \kappa$, there is some accepting run of the original event-clock automaton starting in (l, γ) for some timed word w. Dually, if the accepted language is empty, the underlying event-clock automaton has no accepting run starting in (l, γ) for any $\gamma \in \kappa$ and any w (Lemma 5).

We now describe a procedure that reads timed events and decides whether further events might yield an accepting run (satisfying the formula to check).

The procedure is based on the event-clock automaton as well as the region automaton. It follows the possible *symbolic* computations for the given input along the lines of the event-clock automaton. To decide, whether future events might contribute to an accepting run, the region automaton is consulted.

Let us fix an event-clock automaton $\mathcal{A}_{ec}$ and its region automaton $R(\mathcal{A}_{ec})$ for the moment. Let us consider the timed word $w = (a_0, t_0)(a_1, t_1) \cdots \in T\Sigma^\omega$. Recall that (a_0, t_0) actually means that the first action a_0 occurs at time t_0.

Let Γ_0 be the initial symbolic valuation of $\mathcal{A}_{ec}$ and l_0 one of the initial states of $\mathcal{A}_{ec}$. Now, for the first event (a_0, t_0), we compute the set of successors w.r.t. $\mathcal{A}_{ec}$. If this set is empty, the underlying formula is obviously violated. If not, each successor is a pair (l, Γ). Each (l, Γ) now corresponds to a set of states in the region automaton. If *and only if* for all of them the accepted language is empty, the underlying property is violated, which follows directly from Theorem 2 and Lemma 5. We continue with each successor state (l, Γ) for which a corresponding accepting state of $R(\mathcal{A}_{ec})$ exists, reading the input event.

Thus, the generated procedure keeps a set of possible state-symbolic valuation pairs that represent the possible current states of $\mathcal{A}_{ec}$ (giving credit to the non-deterministic nature of $\mathcal{A}_{ec}$). Furthermore, the transition table of $\mathcal{A}_{ec}$ and the states of $R(\mathcal{A}_{ec})$ enriched with emptiness per state information are stored as look-up tables.

Remark 1. To enhance the practical applicability of our approach, we adjust the procedure slightly: the formal framework described above requires the monitor to complain iff for some prefix $(a_0, t_0) \ldots (a_i, t_i)$ no accepting run exists. In particular, it is assumed that

$$\begin{tikzcd}
\text{Input} & \text{(1) Formula} & \text{(2) ECA} & \text{(3) Region automaton} & \text{(4) Emptiness per state} & \text{(5) Monitor} \\
 & \varphi \arrow[r] & \mathcal{A}_{ec}^{\varphi} \arrow[r] & R^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & \bar{A} \\
 & \neg\varphi \arrow[r] & \mathcal{A}_{ec}^{\neg\varphi} \arrow[r] & R^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Fig. 3. The procedure for getting $[u \models \varphi]$ for a given $\varphi \in \text{TLTL}_3$.

"a watch is consulted only when some action occurs". But the time transitions yielding the subsequent regions in the region automaton actually (often) constrain the possible occurrence of some future event a. For each current valuation Γ corresponding to a set of regions, we check in $R(\mathcal{A})$ the possible accepting time successors and compute a maximal time bound before some event has to occur to reach an accepting state. Thus, in practice, we can set a timer interrupt, when such a bound exists, and decide for rejection, when a timeout occurs before a suitable action has been read.

The overall monitor procedure for TLTL$_3$ is similar to the untimed case and summarised in Fig. 3. However, since we have to consider the region automaton (with emptiness per state information) together with the current clock valuation to compute the timed successor, we do not get an NFA neither can determinise to get a DFA (at least in a straightforward manner). We therefore propose for the overall monitor procedure to rely on $R(\mathcal{A}_{ec}^{\varphi})$ and $R(\mathcal{A}_{ec}^{\neg\varphi})$ in an on-the-fly manner, as described above.

Remark 2. We have used region automata to keep our presentation short and simple. The key property of our monitor construction, however, is *stability* of the region equivalence. Thus, our approach can be improved by taking a coarser stable partition of the underlying timed transition system instead of the region equivalence. Such partitions have been studied extensively in [26].

Complexity. We consider again Fig. 3, and observe that step 1 is constant. The region automaton of $\mathcal{A}_{ec}^{\varphi}$ (resp. $\mathcal{A}_{ec}^{\neg\varphi}$) is exponential with respect to the length of the underlying formula φ as well as the largest constant K appearing in φ. Following the different paths for some prefix (due to the non-determinism of the region automaton) might cause further exponential blow-up in space, in the worst case.

5 Conclusions

The paper presented a monitor construction for (T)LTL formulae. For LTL, we have shown the construction to be optimal, in that no smaller deterministic finite state monitor accepting the same language as ours can be constructed. For both, LTL and TLTL, the construction reflects minimality, such that *true* or *false* is returned by the monitor as early as an observed behavioural trace allows. The latter is an implicit property of the constructed monitor and does not require additional analyses, or data structures besides the monitor itself.

We have already implemented the untimed setting and integrated the monitor generator within a larger logging and unit testing framework. We have examined a standard C++ pitfall and provided a runtime verification solution to this problem, which is efficient in terms of engineering overhead as well as runtime penalty. These examples are available from `http://runtime.in.tum.de/` in a full version of the paper including details of the implementation as well.

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles and Techniques and Tools*. Addison-Wesley, 1986.
2. R. Alur. Timed automata. In *NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems*, 1998.
3. R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
4. R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *TCS*, 211(1-2):253–273, 1999.
5. A. Bauer, M. Leucker, and C. Schallhart. Model-based runtime analysis of distributed reactive systems. In *ASWEC'06*. IEEE, 2006.
6. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In *Hybrid Systems III*, LNCS 1066. Springer, 1996.
7. P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *FoSSaCS*, LNCS 3441. Springer, 2005.
8. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-based Testing of Reactive Systems*, LNCS 3472. Springer, 2005.
9. M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using spin. In *SPIN'01*, LNCS 2057.
10. S. Colin and L. Mariani. *Run-Time Verification*, chapter 18. LNCS 3472. [8], 2005.
11. M. d'Amorim and G. Rosu. Efficient monitoring of omega-languages. In *CAV'05*, LNCS 3576. Springer, 2005.
12. D. D'Souza. A logical characterisation of event clock automata. *Int. Journ. Found. Comp. Sci.*, 14(4):625–639, Aug. 2003.
13. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *CAV'03*, LNCS 2725.
14. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *ASE'01*, IEEE, 2001.
15. J. Håkansson, B. Jonsson, and O. Lundqvist. Generating online test oracles from temporal logic specifications. *STTT*, 4(4):456–471, 2003.
16. K. Havelund and G. Rosu. Monitoring Java Programs with Java PathExplorer. *ENTCS*, 55(2), 2001.
17. K. Havelund and G. Rosu. Monitoring programs using rewriting. In *ASE'01*, IEEE, 2001.
18. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *TACAS'02*, 2002.
19. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *FMSD*, 19(3):291–314, 2001.
20. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, LNCS 3253. Springer, 2004.
21. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*. IEEE, 1977.
22. J.-F. Raskin and P.-Y. Schobbens. State clock logic: A decidable real-time logic. In O. Maler, editor, *HART*, LNCS 1201. Springer, 1997.
23. J.-F. Raskin and P.-Y. Schobbens. The logic of event clocks—decidability, complexity and expressiveness. *JALC*, 4(3):247–286, 1999.
24. V. Stolz and E. Bodden. Temporal Assertions using AspectJ. In *Fifth Workshop on Runtime Verification (RV'05)*. ENTCS. to appear.
25. S. Tripakis. Fault diagnosis for timed automata. In W. Damm and E.-R. Olderog, editors, *FTRTFT*, LNCS 2469. Springer, 2002.
26. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *FMSD*, 18(1):25–68, 2001.
27. M. Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*, LNCS 1043. Springer, 1996.