Monitoring Timed Properties (Revisited)*

Thomas Møller Grosen, Sean Kauffman,
Kim Guldstrand Larsen, and Martin Zimmermann

Aalborg University, Aalborg Denmark
{tmgr,seank,kgl,mzi}@cs.aau.dk

Abstract. In this paper we revisit monitoring real-time systems with respect to properties expressed either in Metric Interval Temporal Logic or as Timed Büchi Automata. We offer efficient symbolic online monitoring algorithms in a number of settings, exploiting so-called zones well-known from efficient model checking of Timed Automata. The settings considered include new, much simplified treatment of time divergence, monitoring under timing uncertainty, and extension of monitoring to offer minimum time estimates before conclusive verdicts can be made.

Keywords: Monitoring · Timed Automata · Metric Temporal Logic

1 Introduction

Runtime monitoring has gained acceptance as a method for formally verifying the correctness of executing systems. Monitoring means to test a sequence of observations of a system against a specification, often written in a formal logic. Monitoring contrasts with static verification methods, like model checking, in that it is computationally easier due to only testing a single system execution. Runtime monitoring may also be applied to black-box systems where details about the environment and design of the monitored system are not required to be known in advance.

Many systems have so-called "extra-functional" requirements that must be expressed with respect to time. These systems, generally called real-time systems, are pervasive in modern life as the controllers of cyber-physical systems. To express requirements with time components, logics such as Metric Temporal Logic (MTL) and derivatives like Metric Interval Temporal Logic (MITL) have been developed that extend the more classical Linear Temporal Logic (LTL) with timing constraints [22,2]. Finite Automata have also been extended with time to form Timed Automata [1]. These formalisms allow the expression of notions such as that "a response should occur within 20 milliseconds of a request."

Monitoring timed properties is possible and several solutions have been proposed, each with their own advantages and drawbacks. In this work, we introduce an efficient solution to the online monitoring problem for timed properties under time divergence. Given a property expressed in MITL and a finite timed

* This paper was partly sponsored by the S4OS Villum Investigator Grant and DIREC Digital Research Center Denmark.

sequence, our method determines if the property is guaranteed to be satisfied or violated by any continuation of that finite sequence. Additionally, the eventual satisfaction or violation of a property considers that any future timing constraints will eventually be settled. Note that online monitoring here contrasts with offline monitoring, where the system is assumed to have terminated and timed properties are interpreted with finite semantics. In online monitoring, liveness properties (e.g., "eventually, plaid shirts will be popular") cannot be violated since there will always be more symbols, while they can be violated in offline monitoring, since the entire sequence is known.

In this paper we revisit the monotoring of real-time properties expressed in the logic MITL. In particular, we offer efficient symbolic online monitoring algorithms for a number of settings. The symbolic approach exploits the fact that properties in MITL may be translated into Timed Büchi Automata (TBA) [9] under a point-wise semantics. Our symbolic approach exploits so-called zones[1], which are used for efficient model checking of Timed Automata [8]. In fact, zones have been exploited in the tool UPPAAL TRON [23] for on-line testing that the behavior of a real-time system conforms to a Timed Automata specification.

In the first setting, we offer a new much simplified way of dealing with time divergence. Time divergence means that, during the infinite behavior of a real-time system, time progresses beyond any finite bound. Time divergence is stated as an assumption by most works in the area because it reflects reality. However, the algorithmic support for time divergence in earlier work seems somewhat underdeveloped.

To understand how time divergence impacts monitor verdicts, consider the property "nothing should be observed after an hour." It should be clear that time divergence guarantees that no infinite sequence will satisfy this property. Because we are interested in online monitoring of properties over infinite timed sequences, the language of the property is empty, and its monitor should evaluate any finite prefix to be in violation of it. Conversely, a monitor that does not compensate for time divergence will register an *unknown* verdict for finite prefixes that do not include observations past the hour mark.

In a second setting, we extend our algorithmic method for monitoring to the setting of timing uncertainty, i.e. a setting where the real-valued time-points of events can only be observed up to a given precision. Finally, in a third and final setting, we refine the algorithmic monitor to offer guaranteed minimum time estimates that must pass until a conclusive verdict can be made.

2 Related Work

Many techniques to monitor timed properties have focused on monitoring logics with finite-word semantics. The first work to introduce timed property monitoring is by Roşu et al. focusing on discrete-time finite-word MTL [31]. Basin et al. proposed algorithms for monitoring real-time finite-word properties in [5] and

[1] also known as DBMs: Difference Bounded Matrices.

compared the differences between different time models. Ulus et al. described monitoring Timed Regular Expressions (TREs) for finite words using a union of two-dimensional zones [32,33].

The most closely related work to ours is that by Bauer et al. in which the authors introduced the classical Three-value LTL (LTL_3) monitor construction and then showed how a similar construction could be used for Timed LTL (TLTL) [6]. Their method transforms a TLTL formula to event-clock automata which are strictly less expressive than Timed Büchi Automata (TBAs), which we support. Their algorithm also differs from ours in being based on the so-called region automata. Though this construction *does* provide a principle monitoring algorithm, the performance of zone-based monitoring algorithms provide an order of magnitude improvement. Finally, their monitoring algorithm does not readily seem to support time divergence. Two more recent works have proposed solutions to the problem of monitoring timed languages specified in MTL. Baldor et al. showed how to construct a monitor for a dense-time MITL formula by constructing a tree of timed transducers [3]. They showed how subsets of MITL could be used to limit the complexity of their technique which requires linear space in the size of the input for the full fragment. Ho et al. split unbounded and bounded parts of a dense-time MITL formula for monitoring, using traditional LTL monitoring for the unbounded parts and permitting a simpler construction for the (finite-word) bounded parts [18]. Unlike the work by Baldor et al., their method is size independent of the input. However, it does require non-elementary blowup of the formula to ensure no unbounded operators appear in the context of a bounded operator. They also monitor bounded parts using a dynamic programming formulation that relies on a maximum bound for the number of events in a time span. Neither the solution by Baldor et al. or Ho et al. address time-divergence.

Crucially, none of the previously mentioned works implement their solutions. On the other hand, some tools have been released for monitoring other timed logics. Basin et al. implemented MonPoly, which can monitor an expressive finite-word (safety) fragment of Metric First-Order Temporal Logic (MFOTL) using discrete time semantics [4]. Bulychev et al. implemented a rewrite-based monitoring algorithm similar to the one proposed in [31] for Weighted MTL in the Uppaal SMC tool [11]. R2U2 is a tool for generating monitors for Field Programmable Gate Arrays (FPGAs) developed by Moosbrugger et al. that supports finite-word MITL properties [26]. Much more recently, Chattopadhyay and Mamouras presented a verified monitor for discrete, past-time (finite word) MITL with quantitative semantics [13]. Some tools also exist to convert timed logics to automata which we will cover in the next section.

3 Preliminaries

We first define some notation used throughout the paper. The set of natural numbers (including zero) is $\mathbb{N}$. The set of real numbers is $\mathbb{R}$ and the set of non-negative real numbers is $\mathbb{R}_{\geq 0}$. The set of Boolean values is $\mathbb{B}$ and the three-valued set of monitor verdicts is $\mathbb{B}_3 = \{\top, ?, \bot\}$. We shall assume that $\mathbb{B}_3$ is equipped

with a partial order where $? \sqsubseteq \top$ and $? \sqsubseteq \bot$. Given a set S the set of all its subsets is denoted 2^S. The cross product of two sets S and T is $S \times T$. Given a sequence σ, σ_i denotes the element at the ith position of σ (where one is the first position) and σ^i denotes the suffix of σ starting at index i. Given two sequences s and t, we write $s \cdot t$ to denote their concatenation.

A timed word over a finite alphabet Σ is a pair $\rho = (\sigma, \tau)$ where σ is a non-empty word over Σ and and τ is a sequence of strictly increasing non-negative real numbers of the same length as σ. Timed words may be finite or infinite where the type of finite timed words is $T\Sigma^*$ and the type of infinite timed words is $T\Sigma^\omega$. We also represent a timed word as a sequence of pairs $(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots$. If $\rho = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots (\sigma_n, \tau_n)$ is a finite timed word, we denote by $\tau(\rho)$ the total time duration of ρ, i.e. τ_n.

Metric Temporal Logic We use the Metric Interval Temporal Logic, MITL, in this work to formalize examples because we can translate it into the TBAs that we use in our monitoring algorithm. Brihaye et al. developed the tool MightyL to translate MITL formulas to TBAs in a compositional manner [9]. Some earlier work implemented algorithms to translate subsets of MITL to TBAs as well. Li et al. proposed and implemented Casaal, a tool to construct deterministic approximations of TBAs from $\text{MTL}_{0,\infty}$ formulas [24,10]. Geilen and Dams implemented an algorithm to produce a deterministic Timed Automaton (TA) for dense-time $\text{MITL}_{\leq}$ (a subset of $\text{MTL}_{0,\infty}$) using an on-the-fly tableau construction that discretizes the time domain and only supports an upper bound [17]. Note that some other works exist that provide algorithms for the translation of MITL or related logics to TBAs, but without providing implementations [16,27].

Let Σ be a finite alphabet. The syntax of MITL formulas over Σ is given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X_I\varphi \mid \varphi\ U_I \varphi$$

where $p \in \Sigma$, and I is a non-singular interval over $\mathbb{R}_{\geq 0}$ with endpoints in $\mathbb{N} \cup \{+\infty\}$. Note that we often write $\sim n$ for $I = \{d \in \mathbb{R} : d \sim n\}$ where $\sim \in \{<, \leq, \geq, >\}$, and $n \in \mathbb{N}$.

The semantics of MITL is defined over infinite timed words. Given such a timed word $\rho = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots \in T\Sigma^\omega$, a position $i \geq 1$, and an MITL formula φ, we inductively define the satisfaction relation $\rho, i \models \varphi$ as follows:

$$\begin{array}{lll}
\rho, i \models p & \textit{if} & p = \sigma_i \\
\rho, i \models \neg\varphi & \textit{if} & \rho, i \not\models \varphi \\
\rho, i \models \varphi \vee \psi & \textit{if} & \rho, i \models \varphi \textit{ or } \rho, i \models \psi \\
\rho, i \models X_I\varphi & \textit{if} & \rho, (i+1) \models \varphi \textit{ and } \tau_{i+1} - \tau_i \in I \\
\rho, i \models \varphi\ U_I \psi & \textit{if} & \exists k \geq i \centerdot\ \rho, k \models \psi, \tau_k - \tau_{i-1} \in I \textit{ and } \forall j \centerdot\ 1 \leq j < k \centerdot\ \rho, j \models \varphi
\end{array}$$

where $\tau_0 = 0$. We write $\rho \models \varphi$ whenever $\rho, 1 \models \varphi$. We also define the standard syntactic sugar: $\textit{true} = p \vee \neg p$, $\textit{false} = \neg \textit{true}$, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi = \neg\varphi \vee \psi$, $F_I\varphi = \textit{true}\ U_I\varphi$, and $G_I\varphi = \neg F_I \neg\varphi$. Given an MITL formula φ, its language $\mathcal{L}(\varphi)$ is the set of all infinite timed words that satisfy φ.

Timed Automata A TBA $\mathcal{A}$ is a six-tuple $(Q, Q_0, \Sigma, C, \Delta, \mathcal{F})$, where Σ is a finite alphabet, Q is a finite set of locations, $Q_0 \subseteq Q$ is a set of initial locations, C is a finite set of clocks, $\Delta \subseteq Q \times Q \times \Sigma \times 2^C \times G(C)$ is a finite set of transitions with $G(C)$ being the type of constraints over C, and $\mathcal{F} \subseteq Q$ is a set of accepting locations. A transition $(q, q', \alpha, \lambda, g)$ is an edge from q to q' on input symbol α where λ is the set of clocks to reset and g is a clock constraint over C. A clock constraint is a conjunction of atomic constraints of the form $c \sim n$, where c is a clock, $n \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. A state of a TBA is a pair (q, v) where q is a location in Q and $v : C \to \mathbb{R}_{\geq 0}$ is a valuation mapping clocks to their values. We say that for any $d \in \mathbb{R}_{\geq 0}$, $v + d$ is the valuation where d is added to all clock values in v.

A run of $\mathcal{A}$ from a starting state (q_0, v_0) is a sequence of steps over a timed word (σ, τ) of the form

$$(q_0, v_0) \overset{(\sigma_1,t_1)}{\longrightarrow} (q_1, v_1) \overset{(\sigma_2,t_2)}{\longrightarrow} (q_2, v_2) \overset{(\sigma_3,t_3)}{\longrightarrow} \cdots$$

where for all $i \geq 1$ there is a transition $(q_{i-1}, q_i, \sigma_i, \lambda_i, g_i)$ such that $v_i(c) = 0$ for all c in λ_i and $v_{i-1}(c) + (t_i - t_{i-1})$ otherwise, and g is satisfied by the valuation $v_{i-1} + (t_i - t_{i-1})$. Given a run r, we denote the set of locations visited infinitely many times by r as $inf(r)$. A run r of $\mathcal{A}$ is accepting if $inf(r) \cap \mathcal{F} \neq \varnothing$. The language of $\mathcal{A}$ from a starting state (q, v), denoted $\mathcal{L}(\mathcal{A}, (q, v))$, is the set of all timed words with an accepting run in $\mathcal{A}$ starting from (q, v). We define the language of $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, to be $\bigcup_q \mathcal{L}(\mathcal{A}, (q, v_0))$, where q ranges over all locations in Q_0 and where $v_0(c) = 0$ for all $c \in C$.

Given two TBAs $\mathcal{A} = (Q, Q_0, \Sigma, C, \Delta, \mathcal{F})$ and $\mathcal{A}' = (Q', Q'_0, \Sigma, C', \Delta', \mathcal{F}')$, their intersection is denoted $\mathcal{A} \otimes \mathcal{A}' = (Q^\otimes, Q_0^\otimes, \Sigma, C^\otimes, \Delta^\otimes, \mathcal{F}^\otimes)$, where

- $Q_\otimes = Q \times Q' \times \{1, 2\}$,
- $Q_0^\otimes = Q_0 \times Q'_0 \times \{1\}$,
- $C^\otimes = C \cup C'$ (we assume they are disjoint),
- $\Delta^\otimes = \Delta_1^\otimes \cup \Delta_2^\otimes$ with
  $\Delta_1^\otimes = \{((q_1, q'_1, 1), (q_2, q'_2, i), \alpha, \lambda \cup \lambda', g \wedge g') : (q_1, q_2, \alpha, \lambda, g) \in \Delta$ and $(q'_1, q'_2, \alpha, \lambda', g') \in \Delta'$ and $i = 2$ if $q_1 \in \mathcal{F}$ else $i = 1\}$ and
  $\Delta_2^\otimes = \{((q_1, q'_1, 2), (q_2, q'_2, i), \alpha, \lambda \cup \lambda', g \wedge g') : (q_1, q_2, \alpha, \lambda, g) \in \Delta$ and $(q'_1, q'_2, \alpha, \lambda', g') \in \Delta'$ and $i = 1$ if $q'_1 \in \mathcal{F}'$ else $i = 2\}$,
- and $\mathcal{F}^\otimes = (\mathcal{F} \times Q' \times \{1\}) \cup (Q \times \mathcal{F}' \times \{2\})$.

We note that $\mathcal{L}(\mathcal{A} \otimes \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$.

4 Monitoring in a Timed Setting

In this section, we describe monitoring and show how it applies in the timed setting. We first, briefly, introduce monitoring in the untimed case and then extend it to the timed case.

Traditionally in Runtime Verification (RV), properties are specified using a temporal logic such as LTL and a monitor is constructed from those properties. A monitor is a kind of program that takes a finite word as an input and returns a

verdict depending on the relationship between the input and the property from which the monitor is constructed. Verdicts are usually of the form *accept* ($\top$), *reject* ($\bot$), or *unknown* (**?**), although larger verdict domains exist to provide more information.

In online monitoring (our interest), the properties specify behaviors over infinite sequences of symbols, or words, while the monitor must interpret those specifications over an ever-growing finite prefix of such an infinite word. The most prevalent solution to this problem is to use a monitor semantics where acceptance or rejection means that the finite word *determines* the property and no future suffix can alter the verdict. In the case where the finite prefix does not determine the property, the monitor outputs an *unknown* verdict and continues.

Monitoring languages of timed infinite words works in much the same way as in the untimed setting. A finite prefix of an infinite timed word is checked to see if it determines the property. If all possible infinite extensions of the prefix result in a word that is included in the monitored property, then the monitor returns the $\top$ verdict. If no possible infinite extensions lead to a word that is included in the monitored property, then the monitor returns the $\bot$ verdict. If extensions exist that could lead to either outcome, then the monitor returns **?** and continues monitoring.

Definition 1 (Monitor verdicts for timed languages). *Given a language of infinite timed words $\phi \subseteq T\Sigma^\omega$ and a finite timed word $\rho \in T\Sigma^*$, the function $\mathcal{V} : 2^{T\Sigma^\omega} \to T\Sigma^* \to \mathbb{B}_3$ evaluates to a verdict with the following definition:*

$$\mathcal{V}(\phi)(\rho) = \begin{cases} \top & \text{if } \rho \cdot \mu \in \phi \text{ for all } \mu \in T\Sigma^\omega, \\ \bot & \text{if } \rho \cdot \mu \notin \phi \text{ for all } \mu \in T\Sigma^\omega, \\ \textbf{?} & \text{otherwise.} \end{cases}$$

Example 1. Consider the bounded response property "whenever a is observed, b should be observed within 30 time units" that is specified by the MITL formula $\varphi = G(a \to F_{\leq 30} b)$ where $\Sigma = \{a, b, c\}$. This property corresponds to the TBA shown in Figure 1. This type of time-bounded leadsto property is very common for real-time systems [25]. It states that some trigger a is followed by a reaction b before the deadline elapses. Note we draw the "sink" state q_3 here for illustrative purposes as we will return to this example later in the paper, but it can be omitted without changing the language of the automaton.

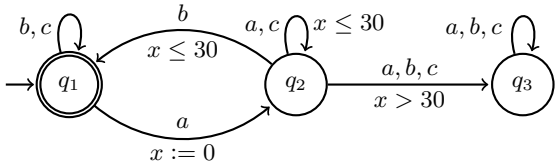


Fig. 1: TBA corresponding to $G(a \to F_{\leq 30} b)$

Now consider the finite timed prefix $\rho_{\text{ok}} = (a, 10), (b, 20)$. The verdict in this case is $\mathcal{V}(\mathcal{L}(\varphi))(\rho_{\text{ok}}) = ?$, since there are infinite extensions of the prefix that satisfy the property and those that violate it. For example, if the pattern of ρ_{ok} was repeated infinitely many times, the resulting infinite timed word $\rho_{\text{ok}}^{\omega} = ((a, 10 \cdot i), (b, 20 \cdot i))^{i \geq 1}$ would satisfy φ. Now suppose the prefix $\rho_{\text{bad}} = (a, 10), (b, 50)$. In this case, the prefix only has infinite extensions that violate the property, so $\mathcal{V}(\mathcal{L}(\varphi))(\rho_{\text{bad}}) = \bot$.

This property demonstrates a way in which timed monitoring differs from the untimed setting. If we remove the time constraint and consider the (unbounded) response property $G(a \rightarrow Fb)$, no finite prefix could ever determine the property and so the only possible verdict would be $?$. This is a classic example of what is called an *unmonitorable* property [29,7,21].

5 Time Divergence

Note that Definition 1 does not take time divergence into account, i.e., a verdict can be based on convergent extensions of a given prefix. As we will see in Example 2, this leads to invalid verdicts. In this section, we consider the monitoring problem in the presence of time divergence and show how it affects the verdicts from monitors for timed languages. Time divergence entails that time will always progress beyond any given time-bound.

We begin by defining the type of infinite time divergent words. These are the only timed words that will occur in practice, since time always diverges. Mathematically, however, the type $T\Sigma^{\omega}$ includes words that are not time divergent, e.g., $(\alpha, \frac{1}{2}), (\alpha, \frac{3}{4}), (\alpha, \frac{7}{8}), \ldots$. The definition states that time divergent words are those where the time sequence is unbounded. Note that we do not consider *finite* timed words either divergent or convergent even though their time sequences technically converge.

Definition 2. *The set of all time divergent words $T_D\Sigma^{\omega} \subseteq T\Sigma^{\omega}$ is the set of all timed words $(\sigma_1, \tau_1), (\sigma_2, \tau_2) \ldots$ such that $\lim_{i \to \infty} \tau_i = +\infty$.*

We now use the set of time divergent words to define a verdict function that accounts for time divergence. Crucially, the properties that we monitor may include non-time divergent words. In that case, the verdict returned by the evaluation function under time divergence $\mathcal{V}_D$ may differ from the verdict returned by $\mathcal{V}$.

Definition 3 (Monitor verdicts under time divergence). *Given a language of infinite timed words $\phi \subseteq T\Sigma^{\omega}$ and a finite timed word $\rho \in T\Sigma^{*}$, the function $\mathcal{V}_D : 2^{T\Sigma^{\omega}} \rightarrow T\Sigma^{*} \rightarrow \mathbb{B}_3$ evaluates to a verdict with the following definition:*

$$\mathcal{V}_D(\phi)(\rho) = \begin{cases} \top & \text{if } \rho \cdot \mu \in \phi \text{ for all } \mu \in T_D\Sigma^{\omega}, \\ \bot & \text{if } \rho \cdot \mu \notin \phi \text{ for all } \mu \in T_D\Sigma^{\omega}, \\ ? & \text{otherwise.} \end{cases}$$

Example 2. Consider the property "the system will continue past 20 time units" represented by the MITL formula $\varphi = F_{\geq 20}a$, where $\Sigma = \{a\}$. This property corresponds to the TBA shown in Figure 2.

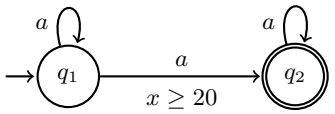


Fig. 2: TBA corresponding to $F_{\geq 20}a$

If this property is monitored, the verdict may change depending on whether time divergence is accounted for. Under time divergence, this property is clearly a tautology since all infinite time divergent words must eventually reach time 20 and location q_2. However, if time divergence is not assumed as in Definition 1, then it is possible for an infinite timed word to never pass time 20 and therefore stay in location q_1.

Suppose, for example, the finite timed prefix $\rho = (a, 10)$. Since φ is a tautology under time divergence, $\mathcal{V}_D(\mathcal{L}(\varphi))(\rho) = \top$. However, $\mathcal{V}(\mathcal{L}(\varphi))(\rho) = ?$, since $T\Sigma^\omega$ contains time-convergent suffixes to ρ that are in not in the language of φ.

To ensure that verdicts are correct for all properties, we monitor an intersection of the given automata with a special TBA that only accepts time divergent words. This TBA, which we will call $\mathcal{A}_D$, is shown in Figure 3. The automaton must visit the left location infinitely often to accept and it can only visit this state once time has passed a threshold of one time unit. Note that the exact threshold is arbitrary and could be any number; the purpose is ensure that the language of the automaton is exactly the language of time divergent words.

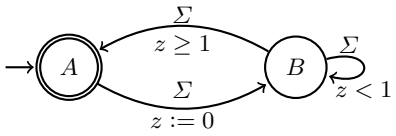


Fig. 3: TBA $\mathcal{A}_D$ to model divergence

Theorem 1. *The language of $\mathcal{A}_D$ is exactly the set of all time divergent words.*

Example 3. We now consider the complement property to Example 2 that is accepted by the MITL formula $\overline{\varphi} = G_{\geq 20}\mathit{false}$. Note that, since we use symbols in our MITL formulas and not propositions, we cannot write $\neg a$ here but must use its complement $\Sigma \setminus \{a\} = \varnothing$ which is equivalent to *false* in our MITL syntax. The TBA for $\overline{\varphi}$ is shown in Figure 4.

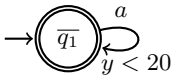


Fig. 4: TBA corresponding to $G_{\geq 20}$*false*

Now we want to intersect this TBA with $\mathcal{A}_D$ to restrict its language to only time divergent words. The result of this operation is shown in Figure 5. Note that we are intentionally showing a trivial example to simplify the presentation; a two state automaton like that in Figure 1 intersected with $\mathcal{A}_D$ has eight locations and, in that case, 23 transitions.

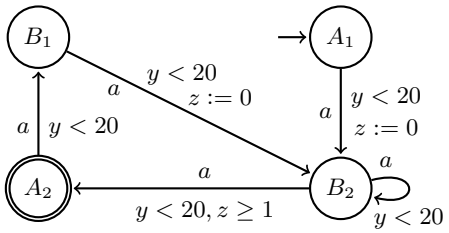


Fig. 5: TBA corresponding to $G_{\geq 20}$*false* intersected with $\mathcal{A}_D$

In this TBA, an accepting run must visit location A_2 infinitely many times. However, it should be clear that this is not possible, since the run must include fewer than 20 combinations of the transition from B_1 to B_2 (which resets the z clock) and the transition from B_2 to A_2, which requires that it passes 1. The reason is that the latter transition also requires the clock y to stay below 20 and y is never reset. The result is that the automaton has an empty language.

6 A Symbolic Method for Monitoring

In this section, we describe an algorithm to monitor languages of infinite timed words that correctly accounts for time divergence. Our algorithm is loosely based on the classical construction for monitoring LTL_3 by Bauer et al. [6], but with alterations to address the many differences between the timed and untimed domains.

Our solution requires that properties are specified as two TBAs - one for the property and one for its complement. Although non-deterministic TBAs are not closed under complementation, we consider this requirement to not be too much of a limitation. This is because we expect a property to be expressed by a user in MITL, which, along with its negation, can be converted to a TBAs using one of the methods described in Section 3.

Before we address the monitoring algorithm, we first introduce the notions of states of a TBA with a non-empty language and state estimates. Given a TBA $\mathcal{A}$, a state (q, v) has a non-empty language when it has an accepting run starting in (q, v).

Definition 4. *Given a TBA $\mathcal{A} = (Q, Q_0, \Sigma, C, \Delta, \mathcal{F})$, the set of states with non-empty language is $S^{ne}_{\mathcal{A}} = \{(q, v) : q \in Q, v \in C \to \mathbb{R}_{\geq 0} \centerdot \mathcal{L}(\mathcal{A}, (q, v)) \neq \varnothing\}$.*

In the following definition, we write $(q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}} (q, v)$ to denote a run that takes $\mathcal{A}$ from (q_0, v_0) to (q, v) processing the input ρ.

Definition 5. *Given a TBA $\mathcal{A}$ and a finite timed word $\rho \in T\Sigma^*$, the set of possible states a run over ρ starting from initial states of $\mathcal{A}$ can end in is given by $\mathcal{T}_{\mathcal{A}}(\rho) = \{(q, v) : (q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}} (q, v)$ where (q_0, v_0) is an initial state of $\mathcal{A}\}$. We call this the state estimate of $\mathcal{A}$ over ρ.*

Note that the state estimate $\mathcal{T}_{\mathcal{A}}(\rho)$ is always a finite set of states of $\mathcal{A}$.

We can now define a function to compute a monitor verdict using Definitions 4 and 5. To determine a verdict for a language $\phi \subseteq T\Sigma^\omega$, the function requires both a TBA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \phi$ and its complement, $\overline{\mathcal{A}}$. Definition 6 states that a finite timed word $\rho \in T\Sigma^*$ positively (negatively) determines the property under time divergence if the states of $\overline{\mathcal{A}} \otimes \mathcal{A}_D$ ($\mathcal{A} \otimes \mathcal{A}_D$) with non-empty languages are disjoint from the state estimate of $\overline{\mathcal{A}} \otimes \mathcal{A}_D$ ($\mathcal{A} \otimes \mathcal{A}_D$) over ρ.

Definition 6 (Monitoring TBAs under time divergence). *Given a TBA $\mathcal{A}$, its complement $\overline{\mathcal{A}}$ ($\mathcal{L}(\overline{\mathcal{A}}) = T\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$), the automaton $\mathcal{A}_D$ such that $\mathcal{L}(\mathcal{A}_D) = T_D\Sigma^\omega$, and a finite timed word $\rho \in T\Sigma^*$, $\mathcal{M} : \mathbb{A} \times \mathbb{A} \to T\Sigma^* \to \mathbb{B}_3$ computes a verdict with the following definition.*

$$\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho) = \begin{cases} \top & \text{if } \mathcal{T}_{\overline{\mathcal{A}} \otimes \mathcal{A}_D}(\rho) \cap S^{ne}_{\overline{\mathcal{A}} \otimes \mathcal{A}_D} = \varnothing \\ \bot & \text{if } \mathcal{T}_{\mathcal{A} \otimes \mathcal{A}_D}(\rho) \cap S^{ne}_{\mathcal{A} \otimes \mathcal{A}_D} = \varnothing \\ ? & \text{otherwise} \end{cases}$$

Theorem 2. *$\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho) = \mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho)$ for all $\rho \in T\Sigma^*$.*

So far, this construction is very similar to the the classical procedure for monitoring LTL_3 with the addition of $\mathcal{A}_D$ to account for time divergence. However, the set of states with non-empty languages of a TBA is likely to be infinite, and its state estimate over a symbolic trace (see Section 7) may be as well. We now present a symbolic online algorithm to compute these infinite sets and their intersections in an efficient manner.

Monitoring Algorithm We assume that the language we will monitor ϕ and its complement $\overline{\phi}$ are given as TBAs $\mathcal{A}_\phi$ and $\mathcal{A}_{\overline{\phi}}$, where $\mathcal{L}(\mathcal{A}_\phi) = \phi$ and $\mathcal{L}(\mathcal{A}_{\overline{\phi}}) = \overline{\phi}$. We begin by computing the intersection of both automata with $\mathcal{A}_D$ (we hereafter refer to these intersection automata as $\mathcal{A}$ and $\overline{\mathcal{A}}$). We continue by finding the states of the automata with non-empty languages, also called the non-empty

states from Definition 4. We then compute the intersection of the non-empty states with the state estimate (from Definition 5) of $\mathcal{A}$ and $\overline{\mathcal{A}}$ over ρ. If one of the intersections is empty, then we can output $\top$ or $\bot$, otherwise, we output $?$.

We can calculate the set $S^{ne}_{\mathcal{A}}$ as a fixpoint using a backwards reachability algorithm. In order to practically work with the states of a TBA we use a symbolic representation of the clock valuations, namely zones. A symbolic state (q, Z) is a pair of a location and a zone. A zone is a finite conjunction of lower and upper bound integer constraints on clocks and clock differences, and may be efficiently represented using so-called Difference Bounded Matrices (DBMs) [8]. We say that $(q, v) \in (q, Z)$ or $v \in Z$ iff $v \models Z$, such that the clock values in v satisfies all constraints in Z. Similarly we say that $v(x) \models Z$ if the value $v(x)$ satisfies the bounds on x in Z. We now define several zone operations, that we will need.

Definition 7. *Given two zones Z and Z' over a set C of clocks, a set of clocks $\lambda \subseteq C$, a positive real number t and an interval $I = [t_1, t_2]$ between two positive real numbers, we define the following operations on zones:*

- $\mathit{free}_\lambda(Z) = \{v : \forall x \in C \centerdot v(x) \models Z \text{ if } x \notin \lambda\}$
- $Z_{\mathit{free}} = \mathit{free}_C(Z)$
- $Z[\lambda] = \{v : \exists v' \in Z \, \forall x \in C \centerdot v(x) = 0 \text{ if } x \in \lambda \text{ otherwise } v(x) = v'(x)\}$
- $Z^{\nearrow} = \{v : \exists v' \in Z \centerdot v = v' + d \text{ for some } d \in \mathbb{R}_{\geq 0}\}$
- $Z^{\searrow} = \{v : \exists v' \in Z \centerdot v = v' - d \text{ for some } d \in \mathbb{R}_{\geq 0}\}$
- $Z^{\nearrow_I} = \{v : \exists v' \in Z \, \forall x \in C \centerdot v'(x) + t_1 \leq v(x) \leq v'(x) + t_2\}$
- $Z \wedge Z' = \{v : v \models Z \text{ and } v \models Z'\}$
- $Z_0 = \{v : \forall x \in C \centerdot v(x) = 0\}$

All of the above operations on zones may be efficiently implemented using the DBM data-structure [8]. We now proceed to develop the online zone-based procedure we use to monitor real-time properties specified by TBAs.

$\mathit{Pred}_{\mathcal{A}}(q, Z)$ (described in Algorithm 1) is the set of symbolic states that can, by a single transition and delay, reach the state (q, Z) of $\mathcal{A}$.

Algorithm 1 Find the predecessors (single transition) of a state

Input: a TBA $\mathcal{A} = (Q, Q_0, \Sigma, C, \Delta, \mathcal{F})$ and a symbolic state (q, Z)
Output: $\mathit{Pred}_{\mathcal{A}}(q, Z)$

```
Predecessors ← ∅
for (q', q, α, λ, g) ∈ Δ do
    Z' = free_λ(Z^↘ ∧ {x = 0 : x ∈ λ} ∧ g)
    Predecessors ← Predecessors ∪ {(q', Z')}
end for
return Predecessors
```

$\mathit{Reach}_{\mathcal{A}}(S)$ (described in Algorithm 2) is the set of symbolic states that can, by at least one transition, reach a state in S.

$\mathit{Reach}^{\infty}_{\mathcal{A}}(Q')$ (described in Algorithm 3) is the set of states that can infinitely many times reach a location in Q'. We can use this to calculate the set of states,

Algorithm 2 Compute the states that can reach the given states with at least one transition

Input: a TBA $\mathcal{A}$ and a set of symbolic states S
Output: $Reach_{\mathcal{A}}(S)$

```
Waiting ← ∅
Passed ← ∅
for s ∈ S do
    Waiting ← Waiting ∪ Pred_A(s)
end for
while Waiting ≠ ∅ do
    select and remove s from Waiting
    Waiting ← Waiting ∪ Pred_A(s)
    Passed ← Passed ∪ {s}
end while
return Passed
```

Algorithm 3 Calculate the set of states that can infinitely often reach a location in Q

Input: a TBA $\mathcal{A}$ and a set of locations Q'
Output: $Reach^{\infty}_{\mathcal{A}}(Q')$

```
S_Q' ← ∅
for q ∈ Q' do
    S_Q' ← S_Q' ∪ {(q, Z_free)}
end for
S_a ← S_Q'
S_b ← ∅
while S_a ≠ S_b do
    S_b ← S_a
    S_a ← Reach_A(S_a ∩ S_Q')
end while
return S_a
```

from which there is a possible accepting run: Given a TBA $\mathcal{A}$ we write $Reach^{\infty}_{\mathcal{A}}$ as a shorthand for $Reach^{\infty}_{\mathcal{A}}(\mathcal{F})$, where $\mathcal{F}$ is the set of accepting locations of $\mathcal{A}$.

Theorem 3. *Given a TBA $\mathcal{A}$. Then $Reach^{\infty}_{\mathcal{A}} = S^{ne}_{\mathcal{A}}$.*

Using this fixpoint, we can do online monitoring given $\mathcal{A}$ and $\overline{\mathcal{A}}$ by storing the state estimate given by a finite timed word over $\mathcal{A}$ and $\overline{\mathcal{A}}$, while continuously checking if the state estimates still overlap with $S^{ne}_{\mathcal{A}}$ and $S^{ne}_{\overline{\mathcal{A}}}$ respectively. If both state estimates still have non-empty languages, then the verdict is *?*, but if all the states in the state estimate of $\mathcal{A}$ have empty languages, then the verdict is $\bot$ ($\top$ for $\overline{\mathcal{A}}$).

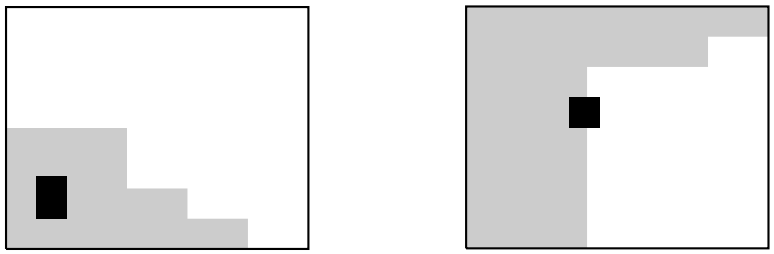
Given the procedure $Succ_{\mathcal{A}}$ described in Algorithm 4 we can compute the state estimate of $\mathcal{A}$ over a finite timed word $\rho \in T\Sigma^*$ of length n iteratively. If S_0 is the set of initial states then $S_n = Succ^{S_{n-1}}_{\mathcal{A}}(\alpha_n, t_n) = \mathcal{T}_{\mathcal{A}}(\rho)$ is the state estimate after ρ.

Algorithm 4 Get the set of possible successor states from S after an input (α, t)

Input: a TBA $\mathcal{A} = (Q, Q_0, \Sigma, C, \Delta, \mathcal{F})$, a set of symbolic states S and a timed input $(\alpha, t) \in \Sigma \times \mathbb{R}_{\geq 0}$
Output: $Succ^S_{\mathcal{A}}(\alpha, t)$

```
Successors ← ∅
for (q, Z) ∈ S do
    for (q, q', α, λ, g) ∈ Δ do
        if Z^↗[t,t] ⊨ g then
            Successors ← Successors ∪ {(q', (Z^↗[t,t] ∧ g)[λ])}
        end if
    end for
end for
return Successors
```

An overview of the online monitoring procedure (see Algorithm 5) with $\mathcal{A}_\phi$ and $\mathcal{A}_{\overline{\phi}}$ is as follows. First we define $\mathcal{A}$ and $\overline{\mathcal{A}}$ as the intersection of each input automaton with $\mathcal{A}_D$. We then use the backwards reachability algorithm to compute the set of states that have a non-empty language (in each TBA). While continuously receiving inputs, we compute the symbolic successor states from the initial states. After each input, we check if there is an overlap between the states with a non-empty language, and the state estimates and output a verdict. The verdict is either $\top$ or $\bot$ when one of the state estimates falls outside the set of states with a non-empty language and *?* otherwise.



(a) State space of $\mathcal{A}$ (b) State spaces of $\overline{\mathcal{A}}$

Fig. 6: Illustration of the state spaces in Algorithm 5 given a timed word ρ. The grey areas are the states with a non-empty language $Reach^\infty_{\mathcal{A}}$ and $Reach^\infty_{\overline{\mathcal{A}}}$. The black areas are the state-estimates after ρ, i.e. $\mathcal{T}_{\mathcal{A}}(\rho)$ and $\mathcal{T}_{\overline{\mathcal{A}}}(\rho)$.

Figure 6 shows an example of the state space, with the non-empty states marked as grey and the current state estimates marked as black. In this example the verdict would be *?*, since both state estimates overlap with the set of states with a non-empty language.

Algorithm 5 Online monitoring procedure given $\mathcal{A}_\phi$ and $\mathcal{A}_{\overline{\phi}}$. Gives a verdict $\top$, $\bot$ or ***?*** after each input

```
time ← 0
A ← A_φ ⊗ A_D
Ā ← A_φ̄ ⊗ A_D
S ← {(q, Z_0) : q is an initial location of A}
S̄ ← {(q, Z_0) : q is an initial location of Ā}
loop
    Receive new input: (α, t) ∈ Σ × ℝ_≥0
    S ← Succ_A^S(α, t − time)                    // Update state estimate
    S̄ ← Succ_Ā^S̄(α, t − time)
    time ← t
    if S ∩ Reach_A^∞ = ∅ then
        output ⊥
    else if S̄ ∩ Reach_Ā^∞ = ∅ then
        output ⊤
    else
        output ?
    end if
end loop
```

Example 4. Consider the MITL formula $\varphi = G(a \rightarrow F_{\leq 30} b)$ from Example 1. We are given the TBA $\mathcal{A}$ in Figure 1 and its complement $\overline{\mathcal{A}}$, the TBA for $\overline{\varphi} = F(a \wedge G_{\leq 30} \neg b)$ shown in Figure 7. As before, we draw the sink state $\overline{q_4}$ here for illustrative purposes. If we were to monitor $\mathcal{L}(\varphi)$, the first step would be to take the intersections $\mathcal{A} \otimes \mathcal{A}_D$ and $\overline{\mathcal{A}} \otimes \mathcal{A}_D$, but we skip that here because of the size of the resulting automata.

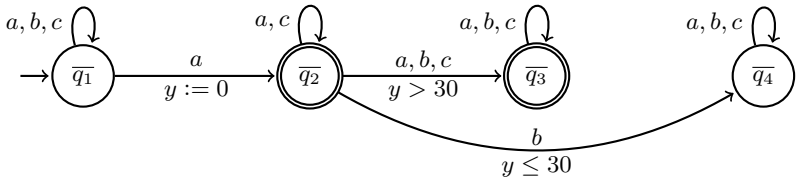


Fig. 7: TBA corresponding to $F(a \wedge G_{\leq 30} \neg b)$

Now suppose the finite prefix $\rho_{\text{ok}} = (a, 10), (b, 20)$ as seen in Example 1. We compute the state estimate for $\mathcal{A}$ as $\mathcal{T}_{\mathcal{A}}(\rho_{\text{ok}}) = \{\ (q_1, \{x = 10\})\ \}$ and for $\overline{\mathcal{A}}$ as $\mathcal{T}_{\overline{\mathcal{A}}}(\rho_{\text{ok}}) = \{\ (\overline{q_1}, \{y = 20\}), (\overline{q_4}, \{y = 10\})\ \}$ where $\{c = v\}$ represents the symbolic constraints on the clocks of the TBAs. Although the algorithm uses a symbolic representation for the state estimates, for a concrete input the clock constraints are equalities. Note that, for $\overline{\mathcal{A}}$, there are two possible states for ρ_{ok} since

$\overline{\mathcal{A}}$ is non-deterministic. For the other Example 1 prefix, $\rho_{\text{bad}} = (a, 10), (b, 50)$, $\mathcal{T}_{\mathcal{A}}(\rho_{\text{bad}}) = \{\ (q_3, \{x = 40\})\ \}$ and $\mathcal{T}_{\overline{\mathcal{A}}}(\rho_{\text{bad}}) = \{\ (\overline{q_1}, \{y = 50\}), (\overline{q_3}, \{y = 40\})\ \}$.

7 Time Uncertainty

So far, we have assumed that we can measure the real-valued time-points τ_i appearing in a finite timed word with arbitrary mathematical precision. Given that our monitor has to be implemented as an algorithm, this cannot necessarily be achieved. To ensure implementability, we rather assume that the time-points are observed with a certain precision. More precisely, we assume that we observe some integer-bounded interval I_i containing τ_i, e.g. $[\lfloor\tau\rfloor, \lceil\tau\rceil]$. In related settings, concrete timing information may be missing for many reasons, particularly when monitoring distributed systems [19,30,28]. The problem we consider is also closely related to the robustness of TA [15] as well as work on monitoring over unreliable channels [20,21].

We assume that during monitoring, we observe a *symbolic* timed word of the form $\rho_s = (\sigma, \upsilon)$, where σ is a finite word over the symbol alphabet Σ and υ is a sequence of time intervals $I_1, I_2, \ldots, I_n$ of the same length as σ. Each time interval I_i is a pair of natural numbers $[l_i, u_i]$ $(l_i < u_i)$ representing a lower and upper bound of the real-valued time-point, when the symbol σ_i occurred. The set of finite symbolic timed words is denoted $T_{\mathbb{S}}\Sigma^*$. We limit bounds in symbolic timed words to natural numbers which is equivalent to supporting rationals with a fixed granularity.

Semantically a symbolic timed word $\rho_s = (\sigma, I_1, I_2, \ldots, I_n)$ covers all timed words (σ, τ) where $\tau_i \in I_i$. We write $\rho \sqsubseteq \rho_s$. Also, whenever $\rho_s = (\sigma, I_1, I_2, \ldots, I_n)$ and $\rho'_s = (\sigma, J_1, J_2, \ldots, J_n)$ are two symbolic timed words, we write $\rho_s \sqsubseteq \rho'_s$ if $I_i \subseteq J_i$ for all $i = 1 \ldots n$.

Now monitoring a language of timed infinite words in the setting of timing uncertainty refines timed monitoring in the following way. Given a finite symbolic prefix, it is checked whether all concrete realization of this prefix determine the property. That is whether all possible infinite extensions of such a concrete realization are included in the monitored property. More formally:

Definition 8 (Monitoring with timing uncertainty). *Given a language of infinite timed words $\phi \subseteq T\Sigma^\omega$ and a finite symbolic timed word $\rho_s \in T_{\mathbb{S}}\Sigma^*$, the function $\mathcal{V}_U : 2^{T\Sigma^\omega} \to T_{\mathbb{S}}\Sigma^* \to \mathbb{B}_3$ evaluates to a verdict with the following definition:*

$$\mathcal{V}_U(\phi)(\rho_s) = \begin{cases} \top & \text{if } \rho \cdot \mu \in \phi \text{ for all } \rho \sqsubseteq \rho_s \text{ and all } \mu \in T_{\mathbb{b}}\Sigma^\omega, \\ \bot & \text{if } \rho \cdot \mu \notin \phi \text{ for all } \rho \sqsubseteq \rho_s \text{ and all } \mu \in T_{\mathbb{b}}\Sigma^\omega, \\ ? & \text{otherwise.} \end{cases}$$

We note that if $\rho_s \sqsubseteq \rho'_s$ then $\mathcal{V}_U(\phi)(\rho_s) \sqsubseteq \mathcal{V}_U(\phi)(\rho'_s)$.

Example 5. Consider the MITL property $F_{[5,6]}a$, and the concrete timed word $\rho = (b, 1.2), (a, 5.4), (c, 7.3)$. Assume that we observe time-points as integer-

bounded intervals of length 1 respectively 2 reflected by the following two symbolic timed words $\rho_s^1 = (b,[1,2]),(a,[5,6]),(c,[7,8])$ and $\rho_s^2 = (b,[1,3]),(a,[5,7]),(c,[7,9])$. Now $\mathcal{V}(F_{[5,6]}a)(\rho) = \top$ and $\mathcal{V}_U(F_{[5,6]}a)(\rho_s^1) = \top$ whereas $\mathcal{V}_U(F_{[5,6]}a)(\rho_s^2) = \bot$.

To obtain a monitoring algorithm for monitoring under timing uncertainty it merely requires that the symbolic successor computation of Algorithm 4 is extended to pairs (α, I) where I is an integer-bounded interval. Thus Algorithm 5 can easily be extended to support time uncertainty.

8 Time Predictive Monitoring

In the timed setting studied in this paper, we want to refine the rather uninformative verdict *?* that occurs during monitoring to provide guaranteed minimum times before a positive or negative verdict can be made.

Example 6. Consider the MITL property $F_{[20,40]}b$. Monitoring the finite timed word $\rho = (a,5.1),(c,21.0),(c,30.4),(b,35.1),(a,40.2)$ will result in three *?* verdicts followed by the verdict $\top$ when $(b,35.1)$ is read. However, we may offer significantly more information, e.g. when reading $(a,5.1)$ it is clear that at least 14.9 time-units must elapse before we can claim that the property holds, and at least 34.9 time-units must elaps before we can claim that it does not hold.

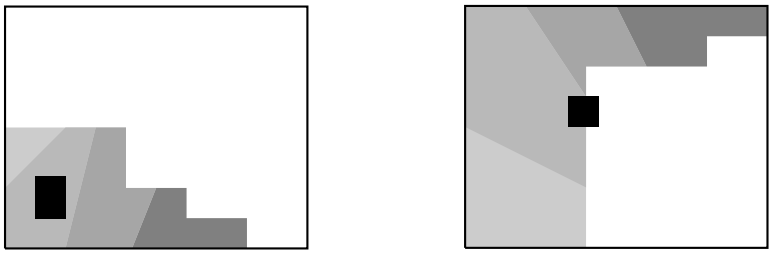
Definition 9 (Time Predictive Monitor Verdicts). *Given a language of infinite timed words $\phi \subseteq T\Sigma^\omega$ and a finite timed word $\rho \in T\Sigma^*$, the function $\mathcal{V}_T : 2^{T\Sigma^\omega} \to T\Sigma^* \to \mathbb{R}^2_{\geq 0}$ evaluates to the verdict $\mathcal{V}_T(\phi)(\rho) = (D^\rho_\phi, D^\rho_{\overline{\phi}})$, where:*

$$D^\rho_\phi = \inf\left\{\tau(\rho') : \rho' \in T\Sigma^* \text{ such that } \forall \mu \in T_b\Sigma^\omega \centerdot \rho \cdot \rho' \cdot \mu \in \phi\right\}$$
$$D^\rho_{\overline{\phi}} = \inf\left\{\tau(\rho') : \rho' \in T\Sigma^* \text{ such that } \forall \mu \in T_b\Sigma^\omega \centerdot \rho \cdot \rho' \cdot \mu \notin \phi\right\}$$

where $\tau(\rho')$ denotes the time duration of ρ'.

The intuition behind Definition 9 is illustrated in Figure 8. We note that when $\mathcal{V}(\phi)(\rho) = \top$ then $\mathcal{V}_T(\phi)(\rho) = (0, +\infty)$. Dually, when $\mathcal{V}(\phi)(\rho) = \bot$ then $\mathcal{V}_T(\phi)(\rho) = (+\infty, 0)$. We note however, that the opposite implications do not hold. As an example consider the property $F_{\geq 5}b$ and the finite timed word $\rho = (c,6)$. Then clearly $\mathcal{V}_T(\phi)(\rho) = (0,+\infty)$ but $\mathcal{V}(\phi)(\rho) \neq \top$.

To make steps toward a time predicting monitoring algorithm, we assume that the property ϕ (as well as $\overline{\phi}$) can be captured by a TBA $\mathcal{A}$. During monitoring, we constantly check whether the state estimate $\mathcal{T}_\mathcal{A}(\rho)$ of the current prefix word ρ intersects the set of states with non-empty language, i.e. $Reach^\infty_\mathcal{A}$. Now $R_\mathcal{A} = Reach_\mathcal{A}(\overline{Reach^\infty_\mathcal{A}})$ describes the set of states that can reach a state with an empty language. By extending the TBA $\mathcal{A}$ with a fresh clock z, the extended set $R^z_\mathcal{A} = Reach_{\mathcal{A}^z}(\overline{Reach^\infty_{\mathcal{A}^z}} \wedge (z = 0))$ captures in a symbolic



(a) State space of $\mathcal{A}$ (b) State space of $\overline{\mathcal{A}}$

Fig. 8: Illustration of the state spaces while monitoring given the inputs $\mathcal{A}$ and $\overline{\mathcal{A}}$ and a timed word ρ. This figure is a refinement of Figure 6. In particular the states with a non-empty language, e.g. $Reach^{\infty}_{\mathcal{A}}$, has been divided into two subsets: the set of states that can reach outside $Reach^{\infty}_{\mathcal{A}}$ (darker grey indicating larger infimum reachability time) and the states that cannot (darkest grey). The black areas are the state-estimates after ρ, i.e. $\mathcal{T}_{\mathcal{A}}(\rho)$ and $\mathcal{T}_{\overline{\mathcal{A}}}(\rho)$.

way the time required to reach a state with an empty language[2]. In particular $d_{\mathcal{A}}(q,v) = \inf\{v_z : (q,v,v_z) \in R^z_{\mathcal{A}}\}$ is the infimum reachability time for a state $(q,v) \in R_{\mathcal{A}}$. For S a set of states, we define $d_{\mathcal{A}}(S)$ to be the supremum of $d_{\mathcal{A}}(q,v)$ over all $(q,v) \in S$. Now we claim the following Theorem, pointing to an effective way of providing guaranteed minimum time predictions of positive and negative verdicts during monitoring.

Theorem 4. *Let $\phi \subseteq T\Sigma^{\omega}$ be a language of infinite timed words. Assume that ϕ is accepted by a TBA $\mathcal{A}_{\phi}$ and the complement $\overline{\phi}$ is accepted by a TBA $\mathcal{A}_{\overline{\phi}}$. Then the following holds:*

$$d_{\mathcal{A}_{\phi}}(\mathcal{T}_{\mathcal{A}_{\phi}}(\rho)) \leq D^{\rho}_{\phi} \quad \textit{and} \quad d_{\mathcal{A}_{\overline{\phi}}}(\mathcal{T}_{\mathcal{A}_{\overline{\phi}}}(\rho)) \leq D^{\rho}_{\overline{\phi}}$$

Moreover, if $\mathcal{A}_{\phi}$ ($\mathcal{A}_{\overline{\phi}}$) is deterministic the first (second) inequality is an equality.

9 Conclusion

In this work, we have revisited the online monitoring problem for timed properties. We presented an efficient online monitoring algorithm for languages of infinite timed words. We require the language and its complement to be expressed as TBAs, a requirement that is, for example, satisfied for languages specified in the logic MITL. Hence our method is applicable in many realistic scenarios.

We showed how to account for time divergence which prior work did not readily seem to support. We also introduced two extensions to our method: support for time uncertainty in the time sequence observed by the monitor, and time predictions that refine the unknown verdict. By supporting time uncertainty

[2] Similar to a method in [12] for time-optimal strategies.

in the observed input, we account for real-world systems where the real-valued time-points of events can only be observed up to a given precision. This limits the soundness of verdicts from monitors that support only concrete timed traces. Time predictions refine the mostly unhelpful unknown verdict to provide extra information on the possible time-to-failure of the monitored system. For nondeterministic TBAs we show how to compute an under-approximation which is exact in the case of deterministic automata. In the future, we will investigate the existence of an exact algorithm for the general case.

While our work was designed to facilitate implementation, this has not yet been completed. Our planned implementation will be distributed in the form of a C++ library with facilities to read TBAs in the UPPAAL XML format [23] that is output by tools like Casaal [24] and MightyL [9]. We also plan to integrate the tool into UPPAAL SMC [14] to replace the current rewrite-based Weighted MTL implementation [11].

Further improvements to our monitoring algorithm include improved support for uncertainties in the input and additional analysis of timed properties. One direction in which we plan to extend the work is to support unobservable symbols in the monitor alphabet. This can be logically extended to consider arbitrary mutations to an input sequence modeled in the form of a TA. We also plan to provide a method to compute the monitorability of a timed property.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM **43**(1) (01 1996)
3. Baldor, K., Niu, J.: Monitoring dense-time, continuous-semantics, metric temporal logic. In: Runtime Verification. pp. 245–259. Springer Berlin Heidelberg (2013). 10.1007/978-3-642-35632-2_24
4. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. LIPIcs, vol. 2, pp. 49–60. Schloss Dagstuhl (2008). 10.4230/LIPIcs.FSTTCS.2008.1740
5. Basin, D., Klaedtke, F., Zălinescu, E.: Algorithms for monitoring real-time properties. In: RV. pp. 260–275. Springer (2012). 10.1007/978-3-642-29860-8_20
6. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science. pp. 260–272. Springer, Berlin, Heidelberg (2006). 10.1007/11944836_25
7. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) **20**(4), 14:1–14:64 (9 2011). 10.1145/2000799.2000800
8. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets, Advances in Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer (2003). 10.1007/978-3-540-27755-2_3

9. Brihaye, T., Geeraerts, G., Ho, H.M., Monmege, B.: MightyL: A compositional translation from MITL to timed automata. In: Computer Aided Verification. pp. 421–440. Springer (2017). 10.1007/978-3-319-63387-9_21
10. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Li, G., Bøgsted Poulsen, D., Stainer, A.: Monitor-based statistical model checking for weighted metric temporal logic. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 168–182. Springer (2012). 10.1007/978-3-642-28717-6_15
11. Bulychev, P., David, A., Larsen, K.G., Legay, A., Li, G., Poulsen, D.B.: Rewrite-based statistical model checking of WMTL. In: Runtime Verification. pp. 260–275. Springer (2013). 10.1007/978-3-642-35632-2_25
12. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR 2005 - Concurrency Theory. LNCS, vol. 3653, pp. 66–80. Springer (2005). 10.1007/11539452_9
13. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Runtime Verification. pp. 383–403. Springer (2020). 10.1007/978-3-030-60508-7_21
14. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Computer Aided Verification. pp. 349–355. Springer (2011)
15. De Wulf, M., Doyen, L., Markey, N., Raskin, J.F.: Robust safety of timed automata. Formal Methods in System Design **33**(1), 45–84 (Dec 2008). 10.1007/s10703-008-0056-7
16. Finkbeiner, B., Kuhtz, L.: Monitor circuits for LTL with bounded and unbounded future. In: Runtime Verification. pp. 60–75. Springer (2009). 10.1007/978-3-642-04694-0_5
17. Geilen, M., Dams, D.: An on-the-fly tableau construction for a real-time temporal logic. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 276–290. Springer (2000). 10.1007/3-540-45352-0_23
18. Ho, H.M., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: Runtime Verification. pp. 178–192. Springer (2014). 10.1007/978-3-319-11164-3_15
19. Jahanian, F., Rajkumar, R., Raju, S.C.V.: Runtime monitoring of timing constraints in distributed real-time systems. Real-Time Systems **7**(3), 247–273 (Nov 1994). 10.1007/BF01088521
20. Kauffman, S., Havelund, K., Fischmeister, S.: Monitorability over unreliable channels. In: International Conference on Runtime Verification (RV'19). LNCS, vol. 11757, pp. 256–272. Springer (2019). 10.1007/978-3-030-32079-9_15
21. Kauffman, S., Havelund, K., Fischmeister, S.: What can we monitor over unreliable channels? International Journal on Software Tools for Technology Transfer **23**, 579–600 (06 2021). 10.1007/s10009-021-00625-z
22. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (Nov 1990). 10.1007/BF01995674
23. Larsen, K.G., Mikucionis, M., Nielsen, B.: Online testing of real-time systems using Uppaal. In: Formal Approaches to Software Testing. LNCS, vol. 3395, pp. 79–94. Springer (2004). 10.1007/978-3-540-31848-4_6
24. Li, G., Jensen, P.G., Larsen, K.G., Legay, A., Poulsen, D.B.: Practical controller synthesis for $\text{mtl}_{0,\infty}$. In: Int. SPIN Symposium on Model Checking of Software. pp. 102–111. SPIN 2017, ACM (2017). 10.1145/3092282.3092303
25. Lindahl, M., Pettersson, P., Yi, W.: Formal design and analysis of a gear controller. In: Steffen, B. (ed.) Tools and Algorithms for Construction and Analysis of Systems (TACAS). LNCS, vol. 1384, pp. 281–297. Springer (1998). 10.1007/BFb0054178

26. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. Formal Methods in System Design **51**(1), 31–61 (Aug 2017). 10.1007/s10703-017-0275-x
27. Ničković, D., Piterman, N.: From mtl to deterministic timed automata. In: Formal Modeling and Analysis of Timed Systems. pp. 152–167. Springer Berlin Heidelberg (2010). 10.1007/978-3-642-15297-9_13
28. Pike, L.: A note on inconsistent axioms in Rushby's "Systematic formal verification for fault-tolerant time-triggered algorithms". IEEE Transactions on Software Engineering **32**(5), 347–348 (May 2006)
29. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods. pp. 573–586. Springer, Berlin, Heidelberg (2006). 10.1007/11813040_38
30. Rushby, J.: Systematic formal verification for fault-tolerant time-triggered algorithms. IEEE Transactions on Software Engineering **25**(5), 651–660 (1999). 10.1109/32.815324
31. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science **113**, 145–162 (2005). 10.1016/j.entcs.2004.01.029, proceedings of the Fourth Workshop on Runtime Verification (RV 2004)
32. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Formal Modeling and Analysis of Timed Systems. pp. 222–236. Springer (2014). 10.1007/978-3-319-10512-3_16
33. Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 736–751. Springer (2016). 10.1007/978-3-662-49674-9_47