Formal Methods in System Design (2026) 69:2
https://doi.org/10.1007/s10703-026-00498-5

RESEARCH

Efficient monitoring of timed properties

Thomas Møller Grosen[1] · **Sean Kauffman**[2] · **Kim Guldstrand Larsen**[1] · **Martin Zimmermann**[1]

Received: 25 June 2025 / Accepted: 15 June 2026

Abstract
In this paper we study monitoring of real-time systems with respect to properties given by a pair of Timed Büchi Automata, one for the property and one for its complement. This includes properties expressible in temporal logics that are closed under complementation and can be translated into Timed Büchi Automata, e.g., Metric Interval Temporal Logic. We introduce efficient symbolic online monitoring algorithms in a number of settings, using difference bound matrices representing zones. Our contributions include a principled treatment of time divergence and monitoring under timing uncertainty. Our online monitoring procedure is implemented in the tool MoniTAal, and shown to effectively monitor properties over long traces.

Keywords Monitoring · Timed automata · Metric interval 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

✉ Thomas Møller Grosen
tmgr@cs.aau.dk

Sean Kauffman
sean.k@queensu.ca

Kim Guldstrand Larsen
kgl@cs.aau.dk

Martin Zimmermann
mzi@cs.aau.dk

[1] Aalborg University, Aalborg, Denmark

[2] Queens University, Kingston, Ontario, Canada

Published online: 31 July 2026

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 [1, 2]. Finite Automata have also been extended with time to form Timed Automata [3]. These formalisms allow the expression of notions such as that "a response shall 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 and a finite timed 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, a request shall be received") 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 study the monitoring of real-time properties expressed by a pair of Timed Büchi Automata (TBA), one for the property and one for its complement. Note that this setting includes properties given in Metric Interval Temporal Logic (MITL), as formulas of MITL can be translated into equivalent TBA [4] under a point-wise semantics. We consider properties over infinite timed words where time is given with infinite precision and its passage is represented by real numbers. Our symbolic approach exploits so-called zones[1], which are used for efficient model checking of Timed Automata [5].

In the first part, we offer a new, simple 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 how time behaves in reality. However, the algorithmic support for time divergence in earlier work on monitoring 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, as time can converge before an hour has passed.

Then, we deal with timing uncertainty, i.e., a setting where the real-valued time-points of events can only be observed up to a given precision. We show that our symbolic monitoring algorithm can easily be adapted to handle timing uncertainty. Finally, we report on a prototype implementation of the presented algorithm and present experiments based on an

[1] Also known as DBMs: Difference Bound Matrices.

industrial case study. We also integrate the formula translation tool MightyPPL [6] to run benchmarks from Timescales [7] in order to compare with the state of the art monitoring tools MonPoly [8] and Reelay [9].

This work is an extended version of an invited contribution presented at FORMATS 2022 [10]. Here, we additionally present more detailed explanations, full proofs, and a prototype implementation.

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 [11]. Basin et al. proposed algorithms for monitoring real-time finite-word properties in [12] and compared the differences between different time models. Ulus et al. described monitoring Timed Regular Expressions (TREs) for finite words using unions of two-dimensional zones [13, 14].

The most closely related work to ours is that by Bauer et al. in which the authors introduced the classical Three-value LTL (LTL3) monitor construction and then showed how a similar construction can be used for Timed LTL (TLTL) [15]. 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. The region construction was introduced as a method to partition the infinite state space of TBA using an equivalence relation between clock assignments [3]. In the region construction, two clock valuations are equivalent if they agree on the integer parts of the clock values and the order of the fractional parts. This forms a finite representation of the infinite state space, but the number of regions is exponential in the parameters of the automaton [3]. Despite this complexity, more compact representations are possible in many cases and can often be achieved using so-called *zones* [16]. In theory, regions can be easier than zones to work with because a region graph is canonical (whereas zones must be normalized) and all valuations within a region are bisimilar. However, in many cases, zone-based algorithms provide an order of magnitude performance improvement over region-based algorithms (see, e.g. [17]).

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 [18]. 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 [19]. 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.

Although the previously mentioned works do not implement their solutions, some tools have been released for monitoring MTL-based 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 [20]. Bulychev et al. implemented a rewrite-based monitoring algorithm similar to the one proposed by Roşu et al. [11] for Weighted MTL in the Uppaal Statistical Model Checking (SMC) tool [21]. R2U2 is a tool for generating monitors for Field Programmable Gate Arrays (FPGAs) developed by Moosbrugger et al. that supports finite-word MITL properties [22]. More recently, Chattopadhyay and Mamouras presented a verified monitor for discrete, past-time (finite word) MITL with quantitative semantics [23]. Ulus introduced a method to monitor past-time (finite word) MTL safety properties over a rational time domain using sequential networks [9] and implemented it in the tool Reelay, which also supports quantified properties.

There is also work on monitoring for other, more expressive timed logics. Monitoring tools for TREs and similar logics are limited to discrete-time such as the AERIAL tool by Basin et al. to monitor metric regular languages using rewriting [24] and Montre by Ulus that monitors TREs using zones [25]. Signal Temporal Logic (STL) adds the complexity of signals to MITL. Nikovic and Maler introduced AMT to monitor STL for past and bounded dense-time properties under a bounded variability assumption [26]. Nikovic and Yamaguchi later extended this work in their RTAMT tool to support STL robustness semantics and add new interfaces [27]. The monitoring algorithms of those tools exploit their bounded-time and bounded-variability assumptions to support conversion to deterministic Timed Automaton (TA) [28]. Havelund and Peled added support to a timed version of their DejaVu tool to monitor a past-time TLTL with quantification over data. They decompose properties into BDDs for each subformula [29]. Henry et al. recently introduced an algorithm to monitor dense-time properties over approximate timed words in a distributed setting, which bears some resemblance to our work on timing uncertainty [30]. However, they restrict the properties they monitor to those representable by deterministic TA, which substantially simplifies the problem. Some tools also exist to convert timed logics to automata which we will cover in the next section.

The topic of monitorability has recently been studied in the timed setting [31, 32]. Monitorability of a property refers to whether or not the property can effectively be monitored i.e., the possibility of deciding satisfaction or violation from a finite prefix.

After the publication of the conference version [10] of this work, the monitoring approach presented here has been extended to a setting with delayed observations [33] and a setting with assumptions and incomplete observations [34].

3 Preliminaries

We first define some notation used throughout the paper. The set of natural numbers (excluding zero) is $\mathbb{N}$, we define $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, the set of rational numbers is $\mathbb{Q}$, the set of non-negative rational numbers is $\mathbb{Q}_{\geq 0}$, the set of real numbers is $\mathbb{R}$, and the set of non-negative real numbers is $\mathbb{R}_{\geq 0}$. The three-valued set of monitor verdicts is $\mathbb{B}_3 = \{\top, ?, \bot\}$. 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 an untimed word over Σ and and τ is a sequence of non-decreasing non-negative real numbers of the same length as σ. Timed words may be finite or infinite where the set of finite timed words is $T\Sigma^*$ and the set of infinite timed words is $T\Sigma^\omega$. We also represent a timed word as a sequence $(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots$ of pairs of letters and non-negative reals. 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 (with the convention $\tau(\varepsilon) = 0$).

If $\rho_1 = (\sigma_1^1, \tau_1^1), (\sigma_2^1, \tau_2^1), \ldots, (\sigma_n^1, \tau_n^1)$ is a finite timed word and $\rho_2 = (\sigma_1^2, \tau_1^2), (\sigma_2^2, \tau_2^2), \ldots$ a finite or infinite timed word and $t \in \mathbb{R}_{\geq 0}$ then the timed word concatenation $\rho_1 \cdot_t \rho_2$ is defined iff $t \geq \tau(\rho_1)$. Then, we define $\rho_1 \cdot_t \rho_2 = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots$ such that

$$\sigma_i = \begin{cases} \sigma_i^1 & \text{if } i \leq n, \\ \sigma_{i-n}^2 & \text{else,} \end{cases} \quad \text{and} \quad \tau_i = \begin{cases} \tau_i^1 & \text{if } i \leq n, \\ \tau_{i-n}^2 + t & \text{else.} \end{cases}$$

3.1 Metric interval 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 [4]. 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 $\mathrm{MITL}_{0,\infty}$ formulas [35, 36]. Geilen and Dams implemented an algorithm to produce a deterministic TA for dense-time $\mathrm{MITL}_{\leq}$ (a subset of $\mathrm{MTL}_{0,\infty}$) using an on-the-fly tableau construction that discretizes the time domain and only supports an upper bound [37]. Note that some other works exist that provide algorithms for the translation of MITL or related logics to TBAs, but without providing implementations [38, 39].

Let Σ be a finite alphabet. The syntax of MITL formulas over Σ is given by the 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 $\mathbb{N}_0 \cup \{+\infty\}$. We often write $\sim n$ for $I = \{d \in \mathbb{R} : d \sim n\}$ where $\sim \in \{<, \leq, \geq, >\}$, and $n \in \mathbb{N}_0$. Note that we consider a pointwise semantics, which is the reason we add the next modality to the syntax. This differs from the original definition of MITL, which did not have a next modality, as it used a continuous semantics (cf. [40], Section 29.6 for a detailed discussion about the differences between the semantics).

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:

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

We write $\rho \models \varphi$ whenever $\rho, 1 \models \varphi$, and say that ρ satisfies φ. Given an MITL formula φ, its language $\mathcal{L}(\varphi)$ is the set of all infinite timed words that satisfy φ.

We also define the standard syntactic sugar $true = p \vee \neg p$, $false = \neg true$, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi = \neg\varphi \vee \psi$, $F_I \varphi = true \; U_I \varphi$, and $G_I \varphi = \neg F_I \neg\varphi$.

3.2 Timed automata

A TBA $\mathcal{A}$ is a six-tuple $(Q, Q_0, \Sigma, \mathcal{X}, \Delta, \mathcal{F})$, where Σ is a finite alphabet, Q is a finite set of locations, $Q_0 \subseteq Q$ is a set of initial locations, $\mathcal{X}$ is a finite set of clocks, $\Delta \subseteq Q \times Q \times \Sigma \times 2^{\mathcal{X}} \times G(\mathcal{X})$ is a finite set of transitions with $G(\mathcal{X})$ being the sets of constraints over $\mathcal{X}$, 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 $\mathcal{X}$. A clock constraint is a conjunction of atomic constraints of the form $x \sim n$, where x is a clock, $n \in \mathbb{N}_0$, and $\sim \in \{<, \leq, =, \geq, >\}$. A state of a TBA is a pair (q, v) where q is a location in Q and $v : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ 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 state (q_0, v_0) is a sequence of steps over an infinite timed word $(\sigma, \tau) \in T\Sigma^\omega$ of the form

$$(q_0, v_0) \xrightarrow{(\sigma_1, \tau_1)} (q_1, v_1) \xrightarrow{(\sigma_2, \tau_2)} (q_2, v_2) \xrightarrow{(\sigma_3, \tau_3)} \cdots$$

where for all $i \geq 1$ there is a transition $(q_{i-1}, q_i, \sigma_i, \lambda_i, g_i)$ such that $v_i(x) = 0$ for all x in λ_i and $v_{i-1}(x) + (\tau_i - \tau_{i-1})$ otherwise, and g is satisfied by the valuation $v_{i-1} + (\tau_i - \tau_{i-1})$, where we use $\tau_0 = 0$. 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 $v_0(x) = 0$ for all $x \in \mathcal{X}$.

Theorem 1 *[2, 4] For each MITL formula φ there exists a TBA $\mathcal{A}$ with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$.*

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

- $Q_\otimes = Q \times Q' \times \{1, 2\}$,
- $Q_0^\otimes = Q_0 \times Q_0' \times \{1\}$,
- $\mathcal{X}^\otimes = \mathcal{X} \cup \mathcal{X}'$ (we assume they are disjoint),

- $\Delta^{\otimes} = \Delta_1^{\otimes} \cup \Delta_2^{\otimes}$ with

$$\Delta_1^{\otimes} = \{((q_1, q_1', 1), (q_2, q_2', i), \alpha, \lambda \cup \lambda', g \wedge g') : (q_1, q_2, \alpha, \lambda, g) \in \Delta \text{ and } (q_1', q_2', \alpha, \lambda', g') \in \Delta' \text{ and } i = 2 \text{ if } q_1 \in \mathcal{F} \text{ else } i = 1\} \text{ and}$$
$$\Delta_2^{\otimes} = \{((q_1, q_1', 2), (q_2, q_2', i), \alpha, \lambda \cup \lambda', g \wedge g') : (q_1, q_2, \alpha, \lambda, g) \in \Delta \text{ and } (q_1', q_2', \alpha, \lambda', g') \in \Delta' \text{ and } i = 1 \text{ if } q_1' \in \mathcal{F}' \text{ else } i = 2\},$$

- and $\mathcal{F}^{\otimes} = Q \times \mathcal{F}' \times \{2\}$.

Lemma 1 *[3]* $\mathcal{L}(\mathcal{A} \otimes \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$.

4 Monitoring in a timed setting

In this section, we describe monitoring and show how it applies in the timed setting. To this end, we first, and briefly, introduce monitoring in the untimed case and then extend it to the timed case, following the approach of Bauer et al. [15].

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. In the timed setting, we also need to know at what time point the prefix ends and when the extension starts. Intuitively, this concatenation time point represents knowledge about the time that has passed since the last observation of the prefix.

Definition 1 (Monitor verdicts for timed languages). Given a language of infinite timed words $\phi \subseteq T\Sigma^{\omega}$, a finite timed word $\rho \in T\Sigma^{*}$, and a time-point $t \in \mathbb{R}_{\geq 0}$ such that $t \geq \tau(\rho)$, the function $\mathcal{V} : 2^{T\Sigma^{\omega}} \to (T\Sigma^{*} \times \mathbb{R}_{\geq 0}) \to \mathbb{B}_3$ evaluates to a verdict with the following definition:

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

If $t < \tau(\rho)$, then $\mathcal{V}(\phi)(\rho,t)$ is undefined.

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 $\varphi = G_{\geq 0}(a \rightarrow F_{\leq 30} b)$ where $\Sigma = \{a, b, c\}$. This property corresponds to the TBA shown in Fig. 1. This type of bounded response property is very common for real-time systems [41]. It states that some trigger a is followed by a reaction b before the deadline elapses. Note that 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.

Now consider the finite timed prefix $\rho = (b, 10), (a, 20)$. The verdict in this case is

$$\mathcal{V}(\mathcal{L}(\varphi))(\rho,t) = \begin{cases} ? & \text{if } t \leq 50, \\ \bot & \text{otherwise.} \end{cases}$$

For $t \leq 50$ there are infinite extensions of the prefix that satisfy the property and those that violate it, e.g., we have

$$\rho \cdot_{40} (b,0),(b,10),(b,20),\ldots = (b,10),(a,20)(b,40),(b,50),(b,60),\ldots \in \mathcal{L}(\varphi)$$

and

$$\rho \cdot_{40} (b,20),(b,30),(b,40),\ldots = (b,10),(a,20)(b,60),(b,70),(b,80),\ldots \notin \mathcal{L}(\varphi).$$

When $t > 50$, then there are no accepting extensions as the a at time 20 has not been followed by a b within 30 time units.

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 [42–44].

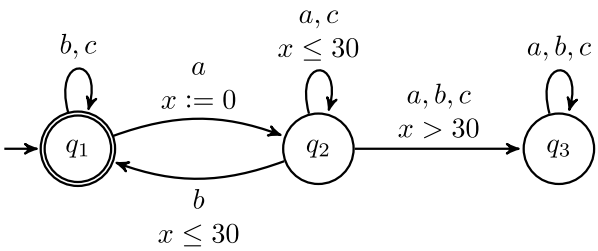

Fig. 1 TBA corresponding to $G_{\geq 0}(a \rightarrow F_{\leq 30} b)$

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 verdicts that are only based on Zeno behaviors. 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. Let us stress here that our algorithms work both in a Zeno and non-Zeno setting.

We begin by defining the set of infinite time-divergent words. These are the only timed words that will occur in practice, since time always diverges. Mathematically, however, the set $T\Sigma^\omega$ 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 $\mathcal{TD}\Sigma^\omega \subseteq T\Sigma^\omega$ 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 $\phi \subseteq T\Sigma^\omega$, a finite timed word $\rho \in T\Sigma^*$, and a time-point $t \in \mathbb{R}_{\geq 0}$ such that $t \geq \tau(\rho)$, the function $\mathcal{V}_D : 2^{T\Sigma^\omega} \to (T\Sigma^* \times \mathbb{R}_{\geq 0}) \to \mathbb{B}_3$ evaluates to a verdict with the following definition:

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

If $t < \tau(\rho)$, then $\mathcal{V}_D(\phi)(\rho, t)$ is undefined.

Example 2 Consider the property "the system will continue past 20 time units after the first observation", represented by the MITL formula $\varphi = F_{\geq 20} a$, where $\Sigma = \{a\}$. This property is captured by the TBA shown in Fig. 2.

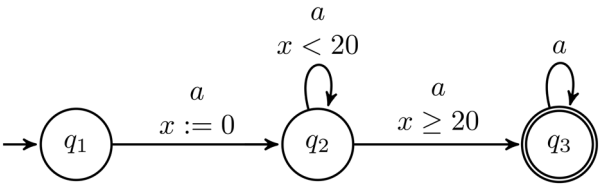

Fig. 2 TBA corresponding to $F_{\geq 20} a$

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 location q_3. However, if time divergence is not assumed as in Definition 1, then it is possible for an infinite timed word to never pass time $x > 20$ after the first observation, and therefore stay in location q_2.

Suppose, for example, the finite timed prefix $\rho = (a, 0), (a, 10)$. Since φ is a tautology under time divergence, $\mathcal{V}_D(\mathcal{L}(\varphi))(\rho, t) = \top$ for all time points $t \geq 10$. However, $\mathcal{V}(\mathcal{L}(\varphi))(\rho, t) = ?$ for all $t < 20$, since $T\Sigma^\omega$ contains time-convergent words μ such that $\rho \cdot_t \mu$ is not in the language of φ, e.g., for $t = 10$ and $\mu = (a, 9), (a, 9.9), (a, 9.99), (a, 9.999), \ldots$ we have $\rho \cdot_t \mu = (a, 0), (a, 10), (a, 19), (a, 19.9), (a, 19.99), (a, 19.999), \ldots \notin \mathcal{L}(\varphi)$.

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 Fig. 3. The automaton must visit the left location infinitely often to accept and it can only visit this location 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 to ensure that the language of the automaton is exactly the language of time-divergent words.

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

We will use the automaton $\mathcal{A}_D$ to restrict the possible extensions of an observation to divergent words.

Example 3 We now consider the complement property to Example 2 that is captured by the MITL formula $\bar{\varphi} = G_{\geq 20} false$. 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 $\bar{\varphi}$ is shown in Fig. 4.

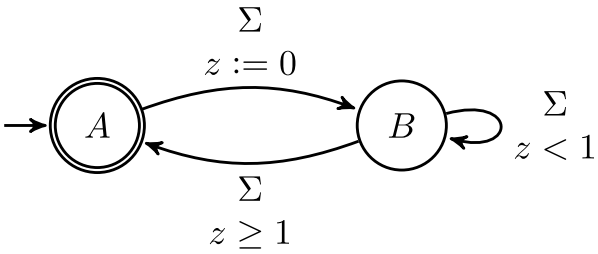

Fig. 3 TBA $\mathcal{A}_D$ capturing time divergence

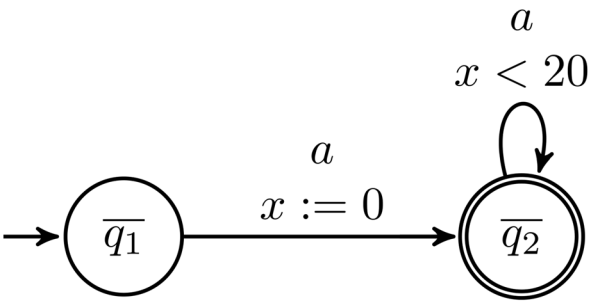

Fig. 4 TBA corresponding to $G_{\geq 20} false$

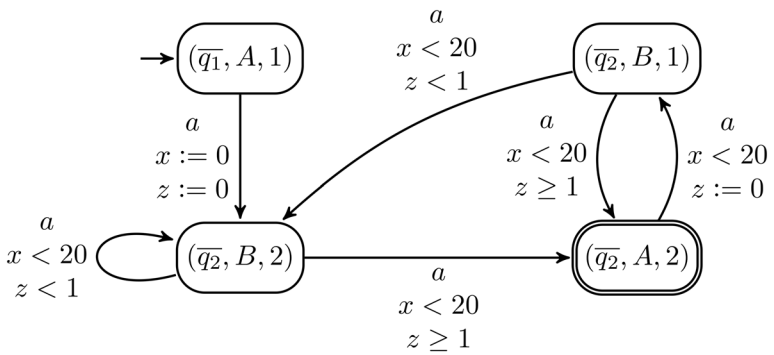

Fig. 5 TBA for the intersection of the TBA from Fig. 4 (corresponding to $G_{\geq 20} false$) and $\mathcal{A}_{\mathrm{D}}$ from Fig. 3 (state names are as described in the construction above Lemma 1). Unreachable locations have been omitted

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 Fig. 5. Note that the intersection automaton contains the unreachable locations $(\overline{q_1}, A, 2)$, $(\overline{q_1}, B, 1)$, $(\overline{q_1}, B, 2)$, and $(\overline{q_2}, A, 1)$, which we omit for simplicity.

In this TBA, an accepting run must visit (the only accepting) location $(\overline{q_2}, A, 2)$ infinitely often. The only transition leaving $(\overline{q_2}, A, 2)$ resets the clock z and all transitions leading to $(\overline{q_2}, A, 2)$ have the guard $z \geq 1$. Hence, at least one unit of time has to pass between any two visits to $(\overline{q_2}, A, 2)$. However, the clock x is never reset and each transition has the guard $x < 20$. Hence, there cannot be an accepting run and the product 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 constructions for monitoring LTL_3 and TLTL by Bauer et al. [15], 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 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 TBAs using one of the methods described in Sect. 3.

Before we address the monitoring algorithm, we first introduce "reach-sets" and "non-empty language states" of a TBA. Given a TBA $\mathcal{A}$, a state (q, v) has a non-empty language when it has an accepting run starting in (q, v), i.e., if $\mathcal{L}(\mathcal{A}, (q, v)) \neq \varnothing$.

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

In the following definition, we write $(q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}} (q_n, v_n)$ for a finite timed word $\rho = (\sigma, \tau) \in T\Sigma^*$ to denote the existence of a finite sequence

$$(q_0, v_0) \xrightarrow{(\sigma_1,\tau_1)} (q_1, v_1) \xrightarrow{(\sigma_2,\tau_2)} \cdots \xrightarrow{(\sigma_n,\tau_n)} (q_n, v_n)$$

of states where for all $1 \leq i \leq n$ there is a transition $(q_{i-1}, q_i, \sigma_i, \lambda_i, g_i)$ such that $v_i(x) = 0$ for all x in λ_i and $v_{i-1}(x) + (\tau_i - \tau_{i-1})$ otherwise, and g is satisfied by the valuation $v_{i-1} + (\tau_i - \tau_{i-1})$, where we use $\tau_0 = 0$. Note that this definition is analogous to the definition of infinite runs in Sect. 3.

Definition 5 Given a TBA $\mathcal{A}$, a finite timed word $\rho \in T\Sigma^*$, and a time-point $t \in \mathbb{R}_{\geq 0}$ with $t \geq \tau(\rho)$, the reach-set of $\mathcal{A}$ over ρ at time t, the set of possible states a run over ρ starting from initial states of $\mathcal{A}$ can end in after time t has passed, is given by

$$\mathcal{T}_{\mathcal{A}}(\rho, t) = \{(q, v + (t - \tau(\rho))) : (q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}} (q, v) \\ \text{where } q_0 \in Q_0 \text{ and } v_0(x) = 0 \text{ for all } x \in \mathcal{X}\}.$$

Remark 2 The reach-set $\mathcal{T}_{\mathcal{A}}(\rho, t)$ is always a finite set of states of $\mathcal{A}$.

We can now define a function for a monitor verdict using Definitions 4 and 5. To determine the 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 $\overline{\mathcal{A}} \otimes \mathcal{A}_{\mathrm{D}}$ ($\mathcal{A} \otimes \mathcal{A}_{\mathrm{D}}$) with non-empty languages are disjoint from the reach-set of $\overline{\mathcal{A}} \otimes \mathcal{A}_{\mathrm{D}}$ ($\mathcal{A} \otimes \mathcal{A}_{\mathrm{D}}$) 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 $\mathcal{A}_{\mathrm{D}}$ such that $\mathcal{L}(\mathcal{A}_{\mathrm{D}}) = \mathcal{TD}\Sigma^\omega$, a finite timed word $\rho \in T\Sigma^*$, and a time-point $t \in \mathbb{R}_{\geq 0}$ such that $t \geq \tau(\rho)$, the function $\mathcal{M} : \mathbb{A} \times \mathbb{A} \to (T\Sigma^* \times \mathbb{R}_{\geq 0}) \to \mathbb{B}_3$ (where $\mathbb{A}$ denotes the set of TBAs) computes a verdict with the following definition:

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

If $t < \tau(\rho)$, then $\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho, t)$ is undefined.

The following theorem shows that the symbolic definition of $\mathcal{M}$ coincides with the definition of $\mathcal{V}_D$ in Definition 3.

Theorem 2 *$\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho, t) = \mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho, t)$ for all $\rho \in T\Sigma^*$ and all $t \in \mathbb{R}_{\geq 0}$ such that $t \geq \tau(\rho)$.*

Proof Let $\mathcal{A}'$ be an arbitrary TBA. We will show that $\mathcal{T}_{\mathcal{A}'}(\rho, t) \cap S^{\mathrm{ne}}_{\mathcal{A}'}$ is nonempty iff there exists a $\mu \in T\Sigma^\omega$ (not necessarily time divergent) with $\rho \cdot_t \mu \in \mathcal{L}(\mathcal{A}')$. Then, we obtain

- $\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho, t) = \top$ iff $\mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho, t) = \top$ by instantiating the equivalence for $\mathcal{A}' = \overline{\mathcal{A}} \otimes \mathcal{A}_{\mathrm{D}}$, and
- $\mathcal{M}(\mathcal{A}, \overline{\mathcal{A}})(\rho, t) = \bot$ iff $\mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho, t) = \bot$ by instantiating the equivalence for $\mathcal{A}' = \mathcal{A} \otimes \mathcal{A}_{\mathrm{D}}$.

This completes the proof, as both functions only have three elements in their codomain and we have shown that two of them have the same preimage w.r.t. both functions.

So, let $\mathcal{T}_{\mathcal{A}'}(\rho, t) \cap S^{\mathrm{ne}}_{\mathcal{A}'} \neq \emptyset$. Then, by definition, there is a state (q, v') of $\mathcal{A}'$ such that

- $(q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}'} (q, v)$ for some $q_0 \in Q_0$ and for v_0 with $v_0(x) = 0$ for all x such that $v' = v + (t - \tau(\rho))$, and
- there is an accepting infinite run of $\mathcal{A}'$ starting in (q, v') that processes some $\mu \in T\Sigma^\omega$.

Let $\rho = (\sigma_1, t_1) \cdots (\sigma_n, t_n)$, which satisfies $t_n = \tau(\rho) \leq t$. Then, due to $(q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}'} (q, v)$, there is a run prefix

$$(q_0, v_0) \xrightarrow{(\sigma_1, t_1)} (q_1, v_1) \xrightarrow{(\sigma_2, t_2)} \cdots \xrightarrow{(\sigma_n, t_n)} (q, v).$$

Also, let $\mu = (\sigma_{n+1}, t_{n+1})(\sigma_{n+2}, t_{n+2}) \cdots$ and let

$$(q, v') \xrightarrow{(\sigma_{n+1}, t_{n+1})} (q_{n+1}, v_{n+1}) \xrightarrow{(\sigma_{n+2}, t_{n+2})} (q_{n+2}, v_{n+2}) \xrightarrow{(\sigma_{n+3}, t_{n+3})} \cdots$$

be the accepting run $\mathcal{A}'$ on μ.

These two runs can be combined into the run

$$\begin{aligned}&(q_0, v_0) \xrightarrow{(\sigma_1, t_1)} (q_1, v_1) \xrightarrow{(\sigma_2, t_2)} \cdots \xrightarrow{(\sigma_n, t_n)} (q, v) \xrightarrow{(\sigma_{n+1}, t_{n+1}+t)} \\ &(q_{n+1}, v_{n+1}) \xrightarrow{(\sigma_{n+2}, t_{n+2}+t)} (q_{n+2}, v_{n+2}) \xrightarrow{(\sigma_{n+3}, t_{n+3}+t)} \cdots\end{aligned}$$

of $\mathcal{A}'$, which starts in (q_0, v_0), processes $\rho \cdot_t \mu$, and is accepting. Hence, $\mathcal{L}(\mathcal{A}')$ is nonempty as required.

Conversely, let there be a $\mu \in T\Sigma^\omega$ with $\rho \cdot_t \mu \in \mathcal{L}(\mathcal{A}')$. Then, there exists an accepting run

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

of $\mathcal{A}'$ starting in some state (q_0, v_0) that processes $\rho \cdot_t \mu$, where $q_0 \in Q_0$ and with $v_0(x) = 0$ for all x. By definition of concatenation, there is an $n \geq 0$ with $t_n \leq t$ and $t_{n+1} \geq t$ (where we use $t_0 = 0$ to allow $n = 0$) such that $\rho = (\sigma_1, t_1) \cdots (\sigma_1, t_n)$ and $\mu = (\sigma_{n+1}, t_{n+1} - t)(\sigma_{n+2}, t_{n+2} - t) \cdots$. The run can be split into two parts:

- The first part is

$$(q_0, v_0) \xrightarrow{(\sigma_1, t_1)} (q_1, v_1) \xrightarrow{(\sigma_2, t_2)} \cdots \xrightarrow{(\sigma_n, t_n)} (q_n, v_n),$$

which witnesses $(q_0, v_0) \xrightarrow{\rho}_{\mathcal{A}'} (q_n, v_n)$. Thus, $(q_n, v_n + (t - t_n)) \in \mathcal{T}_{\mathcal{A}'}(t)$.

- The second part is

$$(q_n, v + (t - t_n)) \xrightarrow{(\sigma_{n+1}, t_{n+1} - t)} (q_{n+1}, v_{n+1}) \xrightarrow{(\sigma_{n+2}, t_{n+2} - t)} (q_{n+2}, v_{n+2}) \xrightarrow{(\sigma_{n+3}, t_{n+3} - t)} \cdots,$$

  which is an accepting infinite run of $\mathcal{A}'$ starting in (q, v') that processes μ.

Hence, $(q_n, v + (t - t_n)) \in \mathcal{T}_{\mathcal{A}'}(\rho, t) \cap S^{\text{ne}}_{\mathcal{A}'}$, which is therefore, as required, nonempty. □

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

6.1 Monitoring algorithm

We assume that the language ϕ we will monitor and its complement $\bar{\phi}$ are given as TBAs $\mathcal{A}_\phi$ and $\mathcal{A}_{\bar{\phi}}$, where $\mathcal{L}(\mathcal{A}_\phi) = \phi$ and $\mathcal{L}(\mathcal{A}_{\bar{\phi}}) = \bar{\phi}$. We begin by computing the intersection of both automata with $\mathcal{A}_\text{D}$ (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 reach-set (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 $S^{\text{ne}}_{\mathcal{A}}$ 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 q and a zone Z. A zone is a finite conjunction of clock constraints on the form $x_1 \sim n$ or $x_1 - x_2 \sim n$, where x_1, x_2 are clocks, $\sim \in \{<, \leq, =, \geq, >\}$, and $n \in \mathbb{Q}_{\geq 0}$. Such a zone describes a convex set of clock valuations.

To compare observed time points with clock constraints in the zones, we add a global clock x_G that will never be reset. Additionally we only consider rational values as inputs and in clock constraints, as we now discuss the implementation of our symbolic algorithm. However, let us stress that a zone still includes all valuations satisfying the constraints, including real-valued clock valuations. Zones may be efficiently represented using so-called Difference Bound Matrices (DBMs) [5].

We now define several zone operations that we will need.

Definition 7 Given a zone Z over a set of clocks $\mathcal{X}$ including the global clock time x_G, a set of clocks $\lambda \subseteq \mathcal{X}$, an interval $I = [t_1, t_2]$ between two positive rational numbers, and a clock constraint g, we define the following operations on zones:

- $free_\lambda(Z) = \{v : \exists v' \in Z_{\bullet}\, \forall x \in \mathcal{X}_{\bullet}\, v(x) = v'(x) if x \notin \lambda\}$
- $Z_{\text{free}} = \text{free}_{\mathcal{X}}(Z)$
- $Z[\lambda] = \{v : \exists v' \in Z\, \forall x \in \mathcal{X}_{\bullet}\, v(x) = 0 \ if x \in \lambda \text{ otherwise } v(x) = v'(x)\}$

- $Z^{\nearrow} = \{v : \exists v' \in Z \bullet v = v' + d \text{ for some } d \in \mathbb{R}_{\geq 0}\}$
- $Z^{\swarrow} = \{v : \exists v' \in Z \bullet v = v' - d \text{ for some } d \in \mathbb{R}_{\geq 0}\}$
- $Z^{\nearrow_I} = Z^{\nearrow} \wedge x_G \in I$
- $Z \wedge g = \{v : v \in Z \text{ and } v \models g\}$
- $Z_0 = \{v : \forall x \in \mathcal{X} \bullet v(x) = 0\}$

Given a set S of symbolic states, $S^{\nearrow_I} = \{(q, Z^{\nearrow_I}) : (q, Z) \in S\}$ is obtained by applying the zone operation $\nearrow_I$ to all zones in the set S.

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

Input: a TBA $\mathcal{A} = (Q, Q_0, \Sigma, \mathcal{X}, \Delta, \mathcal{F})$ and a symbolic state (q, Z)
Output: $Pred_{\mathcal{A}}(q, Z)$
  $Predecessors \leftarrow \varnothing$
  for $(q', q, \alpha, \lambda, g) \in \Delta$ **do**
    $Z' = free_{\lambda}(Z^{\swarrow} \wedge \bigwedge_{x \in \lambda} x = 0) \wedge g$
    $Predecessors \leftarrow Predecessors \cup \{(q', Z')\}$
  end for
  return $Predecessors$

All of the above operations on zones can be efficiently implemented using the DBM data-structure [5].

We now proceed to develop the online zone-based procedure we use to monitor real-time properties specified by TBAs. $Pred_{\mathcal{A}}(q, Z)$ (computed by Algorithm 1) is the set of symbolic states that can, by a single transition and delay, reach the state (q, Z) of $\mathcal{A}$, i.e.,

$$Pred_{\mathcal{A}}(q, Z) = \{(q', Z') : (q', q, \alpha, \lambda, g) \in \Delta \text{ and } Z' = free_{\lambda}(Z^{\swarrow} \wedge \bigwedge_{x \in \lambda} x = 0) \wedge g\}$$

$Reach_{\mathcal{A}}(S)$ (computed by Algorithm 2 implementing a classical backwards analysis (see, e.g. [45])) is the set

$$Reach_{\mathcal{A}}(S) = \{(q, v) : (q, v) \xrightarrow{\rho} (q', v') \text{ s.t.} (q', v') \in S \text{ and } \rho \text{ is nonempty}\}$$

of symbolic states that can, by at least one transition, reach a state in S.

For a TBA $\mathcal{A}$ with set of locations Q and $Q' \subseteq Q$, let $\text{Reach}^{\infty}_{\mathcal{A}}(Q')$ denote the set of states of $\mathcal{A}$ from which an infinite run starts that visits locations from Q' infinitely often. Algorithm 3 computes $\text{Reach}^{\infty}_{\mathcal{A}}(Q')$ which is the limit set $S_n = S_{n-1}$ where

$$S_0 = \{(q, Z_{\text{free}}) : q \in Q'\}$$

is the set of symbolic states whose location is in Q', and where

$$S_i = \text{Reach}_{\mathcal{A}}(S_{i-1} \cap S_0)$$

for $i > 0$ is the set of states that can reach $S_{i-1} \cap S_0$ with at least one transition. Thus, from states in the limit S_n, there is an infinite run that infinitely many times reaches a location in Q'.

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
    s ← Pop(Waiting)
    if s ∉ Passed then
        Waiting ← Waiting ∪ Pred_A(s)
        Passed ← Passed ∪ {s}
    end if
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 Q' of locations of $\mathcal{A}$
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
```

We can use this to calculate the set of states, from which there is an accepting run: Given a TBA $\mathcal{A}$, we write $\mathrm{Reach}^{\infty}_{\mathcal{A}}$ as a shorthand for $\mathrm{Reach}^{\infty}_{\mathcal{A}}(\mathcal{F})$, where $\mathcal{F}$ is the set of accepting locations of $\mathcal{A}$.

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

Proof Consider a TBA with accepting states $S_{\mathcal{F}}$ (i.e., those states with accepting locations) and let $AccPred_{\mathcal{A}}(S) = Reach_{\mathcal{A}}(S) \cap S_{\mathcal{F}}$ be the function mapping a set S of states to its accepting predecessors, i.e., to the set of accepting states that can, by one *or* more transitions, reach a state in S.

Notice that for any set S of states we have $AccPred_{\mathcal{A}}(S) \subseteq S$. The iterative process given by $S^0 = S_{\mathcal{F}}$ and $S^n = AccPred_{\mathcal{A}}(S^{n-1})$ represents part of the transformation in the final loop of Algorithm 3 if $\mathcal{F}$ is given as the parameter. It will converge after a finite number

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

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

```
Successors ← ∅
for (q, Z) ∈ S do
    for (q, q', σ, λ, g) ∈ Δ do
        if Z^{↗[τ,τ]} ⊨ g then
            Successors ← Successors ∪ {(q', (Z^{↗[τ,τ]} ∧ g)[λ])}
        end if
    end for
end for
return Successors
```

of steps to a greatest fixed point S^*. This holds when using zones for the continuous part of the state space, as zones are finite unions of regions [3], which can only be reduced a finite number of times before becoming empty.

The set S^* contains all the states in $S_{\mathcal{F}}$ that can reach a state in S^* by one or more transitions. Thus, we have that S^* is the set of accepting states that can infinitely often visit an accepting state.

Then, $Reach_{\mathcal{A}}(S^*)$ is the set of all states (not necessarily accepting) that can reach accepting states infinitely often, meaning $Reach_{\mathcal{A}}(S^*) = S^{ne}_{\mathcal{A}}$. □

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

Given the procedure $Succ_{\mathcal{A}}$ described in Algorithm 4 we can compute the reach-set of $\mathcal{A}$ over a finite rational-timed word $\rho = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots (\sigma_n, \tau_n) \in T\Sigma^*$ delayed up to a time-point $t \in \mathbb{Q}_{\geq 0}$. If S_0 is the set of initial states and $S_i = Succ^{S_{i-1}}_{\mathcal{A}}(\alpha_i, t_i - t_{i-1})$, then $\mathcal{T}_{\mathcal{A}}(\rho, t) = \{(q, Z^{\nearrow[t-\tau(\rho), t-\tau(\rho)]}) : (q, Z) \in S_n\}$ is the reach-set after observing ρ and time passing up to t.

Termination of Algorithm 1 and Algorithm 4 is straight forward, since they only contain bounded loops. Finally, termination of Algorithm 2 has been shown in the proof of Theorem 3.

Theorem 4 *Algorithm 5 computes $\mathcal{V}_D$ in the following sense: Given a non-empty sequence of inputs in $\in \Sigma \times \mathbb{Q}_{\geq 0} \cup \mathbb{Q}_{\geq 0}$ such that all time points in the sequence are non-decreasing, let ρ be the projection of the input sequence to $\Sigma \times \mathbb{Q}_{\geq 0}$, which is a timed word. Algorithm 5 returns*

- $\mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho, \tau(\rho))$, if the input sequence ends with an input from $\Sigma \times \mathbb{Q}_{\geq 0}$, and
- $\mathcal{V}_D(\mathcal{L}(\mathcal{A}))(\rho, t)$, if the input sequence ends with $t \in \mathbb{Q}_{\geq 0}$.

Proof Let $\rho = (\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)$. An induction shows that when Algorithm 5 is given inputs $(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)$ step-by-step, then S is equal to the reach-set $\mathcal{T}_{\mathcal{A}}(\rho, \tau(\rho))$ and

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

$\mathcal{A} \leftarrow \mathcal{A}_\phi \otimes \mathcal{A}_D$
$\overline{\mathcal{A}} \leftarrow \mathcal{A}_{\overline{\phi}} \otimes \mathcal{A}_D$
$S \leftarrow \{(q, Z_0) : q \text{ is an initial location of } \mathcal{A}\}$
$\overline{S} \leftarrow \{(q, Z_0) : q \text{ is an initial location of } \overline{\mathcal{A}}\}$
loop
  Receive new input: $i \in (\Sigma \times \mathbb{Q}_{\geq 0}) \cup \mathbb{Q}_{\geq 0}$
  if $i = (\sigma, \tau) \in \Sigma \times \mathbb{Q}_{\geq 0}$ **then**
    $S \leftarrow Succ^{S}_{\mathcal{A}}(\sigma, \tau)$
    $\overline{S} \leftarrow Succ^{\overline{S}}_{\overline{\mathcal{A}}}(\sigma, \tau)$
  else if $i = t \in \mathbb{Q}_{\geq 0}$ **then**
    $S \leftarrow S^{\nearrow_{[t,t]}}$
    $\overline{S} \leftarrow \overline{S}^{\nearrow_{[t,t]}}$
  end if
  if $S \cap Reach^{\infty}_{\mathcal{A}} = \emptyset$ **then**
    output $\bot$
  else if $\overline{S} \cap Reach^{\infty}_{\overline{\mathcal{A}}} = \emptyset$ **then**
    output $\top$
  else
    output *?*
  end if
end loop

$\bar{S}$ is equal to the reach-set $\mathcal{T}_{\overline{\mathcal{A}}}(\rho, \tau(\rho))$. Thus, when then given the input t, S is equal to $\mathcal{T}_{\mathcal{A}}(\rho, t)$ and $\bar{S}$ is equal to $\mathcal{T}_{\overline{\mathcal{A}}}(\rho, t)$. Hence, Theorem 3 and Theorem 2 yield the desired result. □

An overview of the online monitoring procedure (see Algorithm 5) with $\mathcal{A}_\phi$ and $\mathcal{A}_{\bar{\phi}}$ 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 reach-sets and output a verdict. The verdict is either $\top$ or $\bot$ when one of the reach-sets falls outside the set of states with a non-empty language and ? otherwise.

Figure 6 shows an example of the state spaces of TBAs $\mathcal{A}$ and $\overline{\mathcal{A}}$, with their non-empty states marked as grey and the reach-sets $\mathcal{T}_{\mathcal{A}}((\sigma_1, \tau_1), \ldots, (\sigma_i, \tau_i))$ and $\mathcal{T}_{\overline{\mathcal{A}}}((\sigma_1, \tau_1), \ldots, (\sigma_i, \tau_i))$ depicted as black circles. As long as both $\mathcal{T}_{\mathcal{A}}((\sigma_1, \tau_1), \ldots, (\sigma_i, \tau_i))$ and $\mathcal{T}_{\overline{\mathcal{A}}}((\sigma_1, \tau_1), \ldots, (\sigma_i, \tau_i))$ contain a (grey) state with non-empty language, the verdict is ?. After three observations, $\mathcal{T}_{\overline{\mathcal{A}}}((\sigma_1, \tau_1), \ldots, (\sigma_3, \tau_3))$ contains no states with non-empty language, i.e., the verdict is $\top$.

Example 4 Consider the MITL formula $\varphi = G_{\geq 0}(a \rightarrow F_{\leq 30} b)$ from Example 1. We are given the TBA $\mathcal{A}$ in Fig. 1 and its complement $\overline{\mathcal{A}}$, the TBA for $\overline{\varphi} = F_{\geq 0}(a \wedge G_{\leq 30} \neg b)$ shown in Fig. 7. As before, we draw the sink state $\overline{q_4}$ here for illustrative purposes. If we

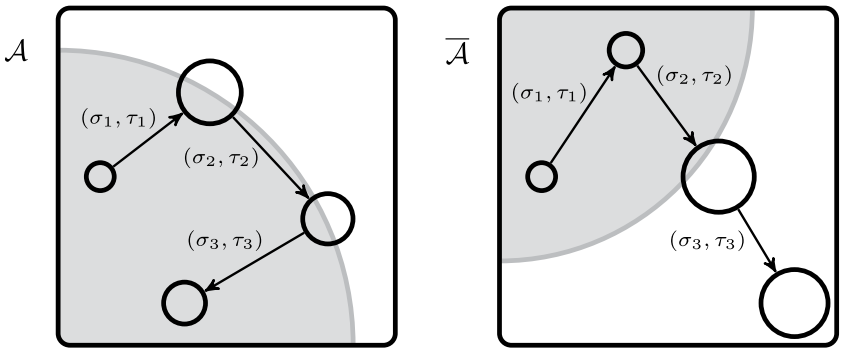

Fig. 6 Evolution of the reach-sets (depicted as circles) in $\mathcal{A}$ (left) and $\overline{\mathcal{A}}$ (right) after 0, 1, 2, and 3 observations. The grey areas are the states with non-empty language, i.e., $Reach^{\infty}_{\mathcal{A}}$ and $Reach^{\infty}_{\overline{\mathcal{A}}}$

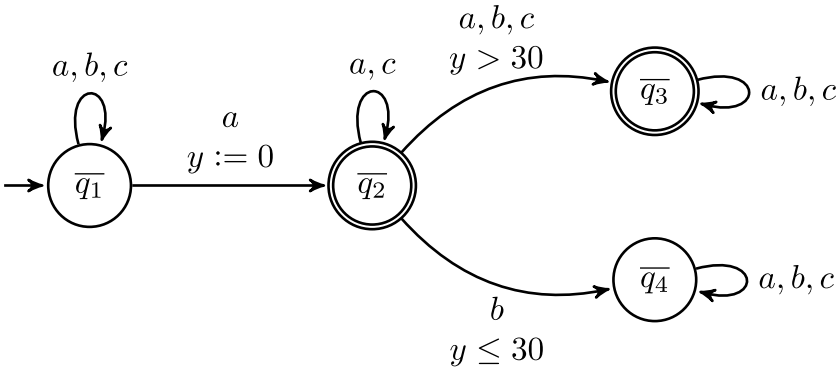

Fig. 7 TBA corresponding to $F_{\geq 0}(a \wedge G_{\leq 30} \neg b)$

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 this step here because of the size of the resulting automata.

First we compute the sets of non-empty language states $S^{ne}_{\mathcal{A}}$ and $S^{ne}_{\overline{\mathcal{A}}}$ as:

- $S^{ne}_{\mathcal{A}} = \{(q_1, true), (q_2, x \leq 30)\}$.
- $S^{ne}_{\overline{\mathcal{A}}} = \{(\bar{q_1}, true), (\bar{q_2}, true), (\bar{q_3}, true)\}$.

Now suppose the finite prefix $\rho = (b, 10), (a, 20)$ as seen in Example 1. We compute the reach-set at time point 25 for $\mathcal{A}$ and $\bar{A}$ as:

- $\mathcal{T}_{\mathcal{A}}(\rho, 25) = \{\ (q_2, x = 5 \wedge x_G = 25)\ \}$.
- $\mathcal{T}_{\overline{\mathcal{A}}}(\rho, 25) = \{\ (\overline{q_1}, y = 25 \wedge x_G = 25), (\overline{q_2}, y = 5 \wedge x_G = 25)\ \}$.

Note that although our algorithm uses a symbolic representation for the reach-sets, for a concrete input the clock constraints are equalities. Also note that, for $\overline{\mathcal{A}}$, there are two possible states for ρ since $\overline{\mathcal{A}}$ is non-deterministic.

Both these reach-sets intersect with the non-empty language states, which means there exist infinite extensions satisfying both properties. However, this is not true if we use a time point greater than 50. For example, the reach-set $\mathcal{T}_{\mathcal{A}}(\rho, 51) = \{\ (q_2, x = 31 \wedge x_G = 51)\ \}$ has an empty intersection with $S^{ne}_{\mathcal{A}}$, thus the verdict would be $\perp$.

7 Timing uncertainty

So far, we have assumed that inputs have rational- rather than real-valued time-points, but in reality it can be real-valued time-points with arbitrary mathematical precision. Instead of approximating a real-valued time-point τ_i as a rational value, we can assume that the time-points are observed with a certain precision. More accurately, we assume that we observe some interval I_i containing τ_i, e.g., $[\lfloor \tau_i \rfloor, \lceil \tau_i \rceil]$. In related settings, concrete timing information may be missing for many reasons, particularly when monitoring distributed systems [46–48]. The problem we consider is also closely related to the robustness of TA [49] as well as work on monitoring over unreliable channels [44, 50].

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 $I_1, I_2, \ldots, I_n$ of time intervals of the same length as σ. Each time interval I_i is a closed interval $[l_i, u_i]$ with integer end-points representing a lower and upper bound of the real-valued time-point at which the symbol σ_i occurred. We require $l_i \leq l_{i+1}$ for all $i < n$. The set of finite symbolic timed words is denoted $\mathcal{IS}\Sigma^*$. Note that, as we are concerned with an algorithmic solution, we limit bounds in symbolic timed words to natural numbers (which is equivalent to supporting rationals with a fixed granularity).

We define the concatenation of symbolic timed words ρ_1 and ρ_2 at time point t where $\rho_1 = (\sigma_1^1, [l_1^1, u_1^1]), (\sigma_2^1, [l_2^1, u_2^1]), \ldots, (\sigma_n^1, [l_n^1, u_n^1])$ is a finite word with $l_n^1 \leq t$, and $\rho_2 = (\sigma_1^2, [l_1^2, u_1^2]), (\sigma_2^2, [l_2^2, u_2^2]), \ldots$ is a finite or infinite word, as $\rho_1 \cdot_t \rho_2 = (\sigma_1, [l_1, u_1]), (\sigma_2, [l_2, u_2]), \ldots$ such that

$$\sigma_i = \begin{cases} \sigma_i^1 & \text{if } i \leq n, \\ \sigma_{i-n}^2 & \text{else,} \end{cases} \quad \text{and} \quad [l_i, u_i] = \begin{cases} [l_i^1, u_i^1] & \text{if } i \leq n, \\ [l_{i-n}^2 + t, u_{i-n}^2 + t] & \text{else.} \end{cases}$$

Semantically, a symbolic timed word $\rho_s = (\sigma, I_1, I_2, \ldots, I_n)$ encodes all timed words (σ, τ) of length n where $\tau_i \in I_i$. In this case, we write $\rho \sqsubseteq \rho_s$.

Remark 3 As we require the lower bounds l_i of a symbolic timed word to be non-decreasing, each symbolic timed word $(\sigma, I_1, I_2, \ldots, I_n)$ encodes at least one concrete timed word, i.e., the word $(\sigma_1, l_1), \ldots, (\sigma_n, l_n)$.

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 $i = 1 \ldots n$.

Example 5 Consider the symbolic timed word $\rho_s = (b, [1, 2]), (a, [5, 6]), (c, [7, 8])$. Then we have $\rho = (b, 1.2), (a, 5.4), (c, 7.3) \sqsubseteq \rho_s$.

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 realizations of this prefix determine the property. That is whether all possible infinite extensions of such a concrete realization are included in the monitored property.

Definition 8 (Monitoring with timing uncertainty). Given a language of infinite timed words $\phi \subseteq T\Sigma^\omega$, a finite symbolic timed word $\rho_s \in \mathcal{TS}\Sigma^*$ ending in $(\sigma, [l, u])$, and a $t \in \mathbb{R}_{\geq 0}$ such that $u \leq t$, the function $\mathcal{V}_U : 2^{T\Sigma^\omega} \to \mathcal{TS}\Sigma^* \times \mathbb{R}_{\geq 0} \to \mathbb{B}_3$ evaluates to a verdict with the following definition:

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

If $u > t$, then $\mathcal{V}_U(\phi)(\rho_s, t)$ is undefined.

Remark 4 If $\rho_s \sqsubseteq \rho_s'$ then $\mathcal{V}_U(\phi)(\rho_s, t) \succeq \mathcal{V}_U(\phi)(\rho_s', t)$, where we define the partial order $\succeq$ over the set $\{\top, ?, \bot\}$ such that $\top \succeq ?$ and $\bot \succeq ?$, but $\top$ and $\bot$ are incomparable. Intuitively, $\succeq$ captures that definitive verdicts are more informative than the inconclusive verdict $?$.

Example 6 Consider the MITL property $F_{[5,6]}a$, expressing that "an a has to occur within five to six time-units after the initial observation", and the concrete timed word $\rho = (b, 0)(b, 1.2), (a, 5.4), (c, 7.3)$. Assume that we observe time-points as integer-bounded intervals of length one respectively length two (except for the initial observation) reflected by the following two symbolic timed words

$$\rho_s^1 = (b, [0, 0]), (b, [1, 2]), (a, [5, 6]), (c, [7, 8])$$

and

$$\rho_s^2 = (b, [0, 0]), (b, [1, 3]), (a, [5, 7]), (c, [7, 9]).$$

Then, we have $\mathcal{V}_D(\mathcal{L}(F_{[5,6]}a))(\rho, 8) = \top$ as well as $\mathcal{V}_U(\mathcal{L}(F_{[5,6]}a))(\rho_s^1, 8) = \top$, whereas $\mathcal{V}_U(\mathcal{L}(F_{[5,6]})a))(\rho_s^2, 8) = ?$, as there are concrete timed words encoded by ρ_s^2 that have an a in the interval $[5, 6]$ and there are concrete timed words encoded by ρ_s^2 that do not have an a in the interval $[5, 6]$, e.g., if the a occurs at time 7.

To obtain a monitoring algorithm for monitoring under timing uncertainty it merely suffices to extend the symbolic successor computation in Algorithm 4 to pairs (α, I) where I is an integer-bounded interval. Since we defined the delay operation for intervals the only necessary change is to replace $Z^{\nearrow[\tau,\tau]}$ with $Z^{\nearrow_I}$ in the innermost *if*-statement of Algorithm 4. Thus, the monitoring procedure (Algorithm 5) can easily be extended to support timing uncertainty.

Theorem 5 *There is an online algorithm that computes* $\mathcal{V}_U$.

8 Tool implementation: MoniTAal

To implement the monitoring algorithms presented in Sects. 6 and 7, we developed the tool MoniTAal[2]. MoniTAal takes two TBAs as input. Each TBA has to be the negation of the other. The user can then either monitor a log of observations, or include the monitor as a library in their own software, and monitor observations in an online fashion. A verdict ($\top$, $\bot$, or ?) is output after each observation.

In this section we demonstrate and benchmark MoniTAal. We monitor properties from a verified gear controller model developed in collaboration between Lindahl et al. [41] and an automotive software company. In addition, we benchmark MoniTAal combined with the formula translation tool MightyPPL, on properties from Timescales.

MoniTAal follows the online monitoring procedure described in Algorithm 5 by first computing the non-empty language states of each input automaton and then in an online fashion, for each observation, updating the reach-set and giving a verdict in $\{\top, \bot, ?\}$. When a verdict $\top$ or $\bot$ is given, the monitoring procedure stops (since no further observations can change the verdict).

8.1 The DBM library Pardibaal

Computing the non-empty language states, as well as the reach-sets from interval-timed traces, requires handling states symbolically using zones. To do this, we implemented the DBM library Pardibaal[3] to handle operations on zones. The existing DBM library UDBM[4] is very efficient and currently used by the model checker Uppaal, but complex to maintain and released under the GPLv3 license. We implemented Pardibaal to have a DBM library under the LGPL licence, permitting inclusion in proprietary applications, as well as to have a modern implementation that is easier to maintain.

Pardibaal implements all standard DBM operations [5] and additional functionality such as subtraction, intersection, and specialized delay operations. In practice, we work with sets of symbolic states and therefore have multiple DBMs (zones). Most of the basic operations are trivially extended to finite unions of zones (i.e., finite sets of DBMs) called federations. Equality or inclusion checking is more complicated using federations, since a single zone could be a subset of a union of multiple zones, without being a subset of any one of them on their own. An exact implementation requires an expensive difference/subtraction operation between DBMs. One way to approximate the relation between federations is to combine the relation between all pairs of DBMs. The subtraction operation is also important in the backwards analysis part used to compute the set of non-empty language states.

8.2 Demonstration of MoniTAal

Consider the property

$$\varphi_a = F_{\leq 10}(a) \wedge G_{\geq 0}(a \rightarrow F_{]0,10]}(a)).$$

[2] https://github.com/DEIS-Tools/MoniTAal.

[3] https://github.com/DEIS-Tools/PARDIBAAL.

[4] https://github.com/UPPAALModelChecker/UDBM.

φ_a requires a to be observed infinitely often, with at least one a being observed within 10 time-units after the first observation, and then the time between two consecutive observations of a is no more than 10. Note that when a has not been observed for more than ten units of time, then the verdict $\bot$ can be given. On the other hand, the verdict $\top$ will never be given when monitoring φ_a.

Now, consider the property

$$\varphi_b = F_{\geq 0}(G_{\geq 0}(\neg b)).$$

φ_b requires that "from some point onwards, we will never see b again". Note that monitoring φ_b will never result in a $\top$ or $\bot$ verdict, since there is always an infinite extension satisfying the property and another one that does not.

In the following example we monitor the property:

$$\varphi_{ab} = \varphi_a \wedge \varphi_b.$$

When monitoring the property φ_{ab} one again obtains the verdict $\bot$ when a has not been observed for more than ten units of time, but the verdict $\top$ is never given. The TBAs accepting the languages of φ_{ab} and $\neg\varphi_{ab}$ are shown in Fig. 8.

During monitoring, the reach-sets are represented as lists $[(q_1, Z_1), (q_2, Z_2), \ldots, (q_k, Z_k)]$ of symbolic states where, for all $i \in [1, k]$, q_i is a location and Z_i is a zone. In our experiments, we want to evaluate the size of the representation of the reach-sets (corresponding to the k). When monitoring a concrete trace (a trace where observation times are singular intervals), each Z_i in the symbolic state encodes a single clock valuation. In addition, if the monitored TBA is deterministic then k will always be 1. Note that in the case of nondeterminism and/or uncertain traces (traces where observation times are intervals) the size of the representation of the reach-set can increase. As previously mentioned, we represent zones as DBMs, but when monitoring concrete traces we represent the clock valuation as a list of assignments rather than a DBM.

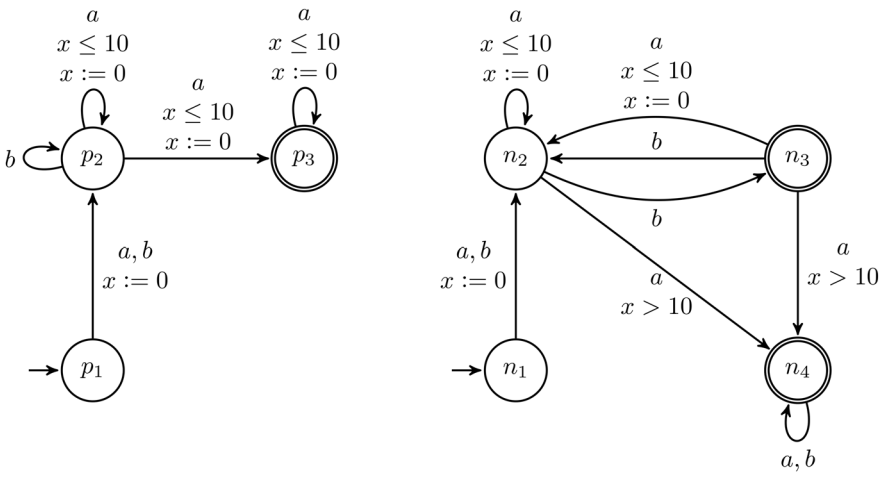

Fig. 8 TBAs accepting the languages of φ_{ab} (left) and $\neg\varphi_{ab}$ (right)

We say that the reach-set representation $[(q_1, Z_1), (q_2, Z_2), \ldots, (q_k, Z_k)]$ is reduced w.r.t. zone-inclusion, if whenever $q_i = q_j$ for $i \neq j$, then $Z_i \not\subseteq Z_j$ and $Z_j \not\subseteq Z_i$.

Example 7 Let us monitor the automata from Fig. 8 over a concrete trace $\rho = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_n, \tau_n)$ containing a series of a's and b's using the reduced state space representation. Looking at just the automaton for φ_{ab}, the reach-set $\mathcal{T}_{\mathcal{A}_{\varphi_{ab}}}(\rho, \tau_n)$ has one of the following three forms:

1. $\emptyset$ if the distance between two a's is greater than 10, or
2. $\{(p_0, \{x = 0, x_G = \tau_n\}), (p_1, \{x = 0, x_G = \tau_n\})\}$ if $\sigma_n = a$, or
3. $\{(p_0, \{x = \tau_n - \tau_{n-1}, x_G = \tau_n\})\}$ if $\sigma_n = b$.

Now we will demonstrate MoniTAal monitoring φ_{ab} over short traces. Figure 9 shows an execution of the trace $(a, 0), (a, 4), (b, 20)$. The trace is entered incrementally on lines 9, 17, and 26. The outputs between the inputs are the reach-sets (as long as the verdict is ?) or a definitive verdict is given, at which point the algorithm terminates. The "positive" and "negative" reach-sets refer to the reach-set of the TBA accepting the language of the monitored property and its negation. The representation of the reach-set is printed as a location with a valuation for all clocks where `0` is the "zero clock" and `global` is the global clock previously referred to as x_G.

We see that the reach-sets of the two automata together contain three states after observing $(a, 4)$ on line 17, which is the maximum size of these sets in this execution. After observing $(b, 20)$ on line 26 the property can no longer be satisfied, so the monitoring ends with a $\bot$ verdict printed as `NEGATIVE`.

Figure 10 is a similar example, except that the trace has uncertain timing. The trace monitored here is $(a, [0, 0]), (a, [0, 10]), (a, [1, 11]), (b, [30, 30])$. The clock values are now handled symbolically as zones, instead of just as single valuations. A zone here is printed as a DBM, where rows and columns refer to clocks. The first row and column is the "zero clock" (the value of which is always zero), the second is the clock x from the automaton, and the third is the global clock (`global`/x_G). An entry $(\prec, n)$ on row i and column j of a DBM is interpreted as the constraint $x_i - x_j \prec n$ where $\prec \in \{<, \leq\}$ and n is an integer. For example, on line 29 one can see that the two first entries of that row indicate the following constraints: $x_G - 0 \leq 10$ and $x_G - x \leq 10$.

Notice that after observing $(a, [20, 30])$ (line 52) the negative reach-set includes two symbolic states in n_4, where the difference between them are clock constraints on x. It is actually unnecessary to store clock constraints on x in this location, since the value of x will at no point in the future be checked by a guard. Because of this, we could ignore the value of this clock, so that all states in n_4 become equivalent.

This is an application of the inactive clock-abstraction by Daws and Yovine [51]. By analysing the automaton, we can find the set of active clocks for each location. A clock is active if there is a path where the clock appears in a guard before being reset. If a clock is not active, it can be ignored. This allows us to prune some symbolic states that, even though they have different clock constraints, will not impact the result. Note that we can never ignore the zero or global clock, because they are actively used in the procedure.

Following the example, it is obvious that the complexity highly depends on the monitored property (and how concise the property automaton is constructed) as well as the obser-

```
$ ./monitaal-bin <...>

Positive reach-set:
p1 : 0 = 0, x = 0, global = 0

Negative reach-set:
n1 : 0 = 0, x = 0, global = 0

Next event: @0 a

Positive reach-set:
p2 : 0 = 0, x = 0, global = 0

Negative reach-set:
n2 : 0 = 0, x = 0, global = 0

Next event: @4 a

Positive reach-set:
p2 : 0 = 0, x = 0, global = 4
p3 : 0 = 0, x = 0, global = 4

Negative reach-set:
n2 : 0 = 0, x = 0, global = 4

Next event: @20 b
Monitoring ended, verdict is: NEGATIVE
Monitored 3 events
```

Fig. 9 Example of monitoring the property φ_{ab} over the concrete timed trace $(a, 0), (a, 4), (b, 20)$

vations. For the simple response property seen in Example 1, the time and memory usage per observation is negligible (below 100 microseconds). This is mainly because the property automata are small, and the inactive clock-abstraction implies an upper bound on the size of the representation of the reach-sets in those cases. Since we are doing online monitoring, the previous observations do not take up space in memory. The response time between verdicts can increase more when the representation of the reach-set becomes large. We say that the response time is the time it takes between providing an observation to receiving a verdict.

To demonstrate monitoring properties and providing verdicts in a more realistic scenario we will examine the gear controller model by [41].

The gear controller model is separated into several components, one of which is the interface. The interface sends requests *ReqNewGear* after which the gear controller changes the gear and sends a response *NewGear*. The expected time it takes to change gear depends

```
$ ./monitaal-bin <...>
...

Next event: @[0,0] a

Positive reach-set:
p2
<<<<<<
<=0      <=0      <=0
<=0      <=0      <=0
<=0      <=0      <=0
>>>>>>

Negative reach-set:
n2
<<<<<<
<=0      <=0      <=0
<=0      <=0      <=0
<=0      <=0      <=0
>>>>>>

Next event: @[0, 10] a

Positive reach-set:
p2
<<<<<<
<=0      <=0      <=0
<=0      <=0      <=0
<=10     <=10     <=0
>>>>>>
p3
<<<<<<
<=0      <=0      <=0
<=0      <=0      <=0
<=10     <=10     <=0
>>>>>>

Negative reach-set:
n4
<<<<<<
<=0      <-10     <-10
<=15     <=0      <=0
<=15     <=0      <=0
>>>>>>
n2
<<<<<<
<=0      <=0      <=0
<=0      <=0      <=0
<=10     <=10     <=0
>>>>>>

Next event: @[20, 30] a

Positive reach-set:
p2
<<<<<<
<=0      <=0      <=-20
<=0      <=0      <=-20
<=20     <=20     <=0
>>>>>>
p3
<<<<<<
<=0      <=0      <=-20
<=0      <=0      <=-20
<=20     <=20     <=0
>>>>>>

Negative reach-set:
n4
<<<<<<
<=0      <=-20    <=-20
<=30     <=0      <=0
<=30     <=0      <=0
>>>>>>
n4
<<<<<<
<=0      <-10     <=-20
<=30     <=0      <=0
<=30     <=10     <=0
>>>>>>
n2
<<<<<<
<=0      <=0      <=-20
<=0      <=0      <=-20
<=20     <=20     <=0
>>>>>>

Next event: @[41, 45] b
Monitoring ended, verdict is: NEGATIVE
Monitored 4 events
```

Fig. 10 Monitoring the property φ_{ab} over the uncertain timed trace$(a, [0, 0]), (a, [0, 15]), (a, [20, 30]), (b, [41, 45])$

on whether or not the gear is being changed to or from neutral, and if some non-critical failures are observed in the engine. To describe this we take the conjunction of two properties that distinguish between gear-change requests between neutral and non-neutral gears. The request $ReqNewGear_{\text{neu}}$ is used when the gear change is to or from neutral, while $ReqNewGear_k$ is used otherwise. $error_1$ and $error_2$ refer to observations in the engine signaling problems with matching torque and speed.

$$\begin{aligned}\varphi_{gear-neu} = G_{\geq 0}(ReqNewGear_{neu} \rightarrow (X_{[150,900]}(NewGear)) \\ \vee (X_{[150,900]}(error_1) \wedge F_{[550,1055[}(NewGear)) \\ \vee (X_{[150,900]}(error_2) \wedge F_{[450,1205]}(NewGear)))\end{aligned}$$

$$\begin{aligned}\varphi_{gear-k} = G_{\geq 0}(ReqNewGear_k \rightarrow (X_{[400,900]}(NewGear)) \\ \vee (X_{[150,900]}(error_1) \wedge F_{[700,1055[}(NewGear)) \\ \vee (X_{[150,900]}(error_2) \wedge F_{[750,1205]}(NewGear)))\end{aligned}$$

The combined property $\varphi_{\text{gear}} = \varphi_{\text{gear-neu}} \wedge \varphi_{\text{gear-k}}$ describes the timing guarantees of the gear controller when changing gears. To monitor the property, we simulate the behavior of the gear controller. Starting in neutral, a request is sent to change the gear up or down. A response is observed at some time point, depending on whether or not a non-critical error occurred.

For this demonstration we simulate a gear controller that can change gears 150 time units earlier or later than the specification allows. The simulation selects the timing of the gear change uniformly. Other than the timing, the gear controller behaves correctly. Furthermore, the behavior is observed with a timing uncertainty, meaning that all timing information of observations are intervals of 100 time units. Note that the simulation of the faulty gear controller is not guaranteed to exhibit a faulty gear change at any point.

Figure 11 shows a histogram reporting how many times a gear change was observed, before an error was detected, across 1000 simulations of the faulty gear controller. As expected, the chance of no error occurring decreases. The verdict given in all these cases are simply $\perp$, since satisfaction of the property φ_{gear} cannot be determined by a finite prefix. The demonstration shows that in all 1000 cases an error was detected. The longest trace showed an error after 65 gear changes.

8.3 Performance

We run experiments of MoniTAal on the property φ_{ab} described above, as well as the gear controller property φ_{gear} taken from [41]. For these experiments the properties φ_{gear} and φ_{ab} are monitored over traces of three different lengths, and are monitored with combinations of timing uncertainty and under time divergence.

Table 1 shows the experimental results for monitoring the property φ_{ab}. The results are shown for two different traces of (parametric) length k:

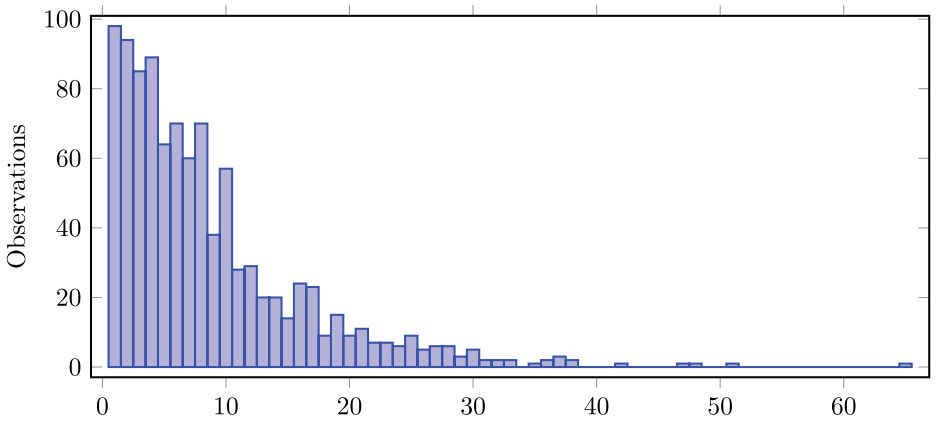

Fig. 11 Histogram of the number of gear changes observed before a fault was detected over 1000 runs

Table 1 Results for monitoring φ_{ab}. Here, "total time" refers to to the overall runtime of the algorithm, "Max response" refers to the maximal time between a new observation and the corresponding verdict, and "Max size" refers to the maximal size of the reach-sets of both automata together. The results are the mean values of 50 runs.

Word length	Total time	Max response	Max Size
ρ_c			
25,000	43.0 ± 0.6 ms	25.9 ± 8.1 μs	3
50,000	86.0 ± 0.9 ms	31.1 ± 9.0 μs	3
100,000	172.3 ± 3.1 ms	33.2 ± 7.7 μs	3
200,000	343.4 ± 2.0 ms	38.2 ± 5.2 μs	3
ρ_c w. time divergence			
25,000	105.8 ± 0.9 ms	25.2 ± 6.6 μs	4
50,000	211.8 ± 2.8 ms	30.3 ± 11.2 μs	4
100,000	423.1 ± 4.0 ms	30.4 ± 8.4 μs	4
200,000	849.1 ± 11.2 ms	33.5 ± 7.0 μs	4
ρ_u			
25,000	150.9 ± 0.6 ms	31.3 ± 7.8 μs	3
50,000	301.8 ± 1.9 ms	30.3 ± 7.2 μs	3
100,000	605.0 ± 5.9 ms	33.9 ± 5.4 μs	3
200,000	1,208.0 ± 4.3 ms	36.3 ± 7.8 μs	3
ρ_u w. time divergence			
25,000	2,959.2 ± 16.9 ms	166.8 ± 6.4 μs	16
50,000	5,926.7 ± 40.6 ms	170.6 ± 8.0 μs	16
100,000	11,861.5 ± 47.0 ms	170.8 ± 5.5 μs	16
200,000	23,760.1 ± 123.8 ms	173.2 ± 8.1 μs	16

1. $\rho_c = (a, 1), (a, 2), \ldots, (a, k-1), (b, 30+k)$
2. $\rho_u = (a[0, 10]), (a, [1, 11]), \ldots, (a, [k-1, k+9]), (b, [k+30, k+30])$

ρ_c is a concrete trace while ρ_u is an uncertain trace.

The results show us that we, in all cases, obtain a bound on the size of the reach-set. This is very useful, since the size of the reach-set is what primarily determines the response time. However, the maximum number of *different* symbolic states is determined by the construction of the TBAs. This can be observed by looking at the impact of monitoring under time divergence in the case of uncertain timing. When monitoring under time divergence we use TBA-intersection, resulting in much larger TBAs, thus increasing the maximum number of possible symbolic states.

The observed trace can also affect the size of the reach-set, because observations can enable a different number of transitions. This can either be because of non-determinism, or because of uncertain timing as seen in the difference between the result of ρ_c and ρ_u.

Table 2 shows the results of monitoring the gear controller properties in scenarios similar to the φ_{ab} experiment. In this case the maximum reach-set size does not change when monitoring a concrete trace under time divergence. This is because both the property automata and the time divergence automaton are deterministic. Thus under intersection, the automaton is still deterministic. Monitoring a deterministic automaton with precise/concrete observations will always result in a singleton reach-set. However timing uncertainty does increase the size of the reach-set, and even more so when also combined with time divergence.

Table 2 Results for monitoring the φ_{gear} over a simulation of the gear controller model from [41]. Here, "total time" refers to to the overall runtime of the algorithm, "Max response" refers to the maximal time between a new observation and the corresponding verdict, and "Max size" refers to the maximal size of the reach-sets of both automata together. The results are the mean values of 50 runs.

Word length	Total time	Max response	Max Size
Gear Controller			
1000	4.1 ± 0.0 ms	14.9 ± 9.3 μs	2
5000	20.4 ± 0.5 ms	21.4 ± 9.2 μs	2
10000	40.4 ± 0.5 ms	20.7 ± 5.3 μs	2
Gear Controller w. time divergence			
1000	9.8 ± 0.1 ms	25.3 ± 7.6 μs	2
5000	49.0 ± 0.8 ms	28.8 ± 6.7 μs	2
10000	97.6 ± 0.6 ms	30.3 ± 6.6 μs	2
Gear Controller w. uncertainty			
1000	6.8 ± 0.1 ms	18.8 ± 6.7 μs	3
5000	34.0 ± 0.3 ms	22.6 ± 4.7 μs	3
10000	68.1 ± 0.6 ms	22.5 ± 5.7 μs	3
Gear Controller w. uncertainty and time divergence			
1000	41.2 ± 0.4 ms	81.1 ± 2.9 μs	12
5000	209.5 ± 1.9 ms	87.5 ± 6.1 μs	13
10000	421.4 ± 4.9 ms	88.2 ± 5.4 μs	13

8.4 Time divergence

Running the tool on the property $F_{\geq 20} a$ from Example 2 illustrates how intersection with the time divergence automaton can give an immediate verdict. Figure 12 shows MoniTAal being run with and without time divergence (using option `-div <alphabet>`). As expected, a verdict is provided instantly only when time divergence is taken into account.

8.5 Logic to automata translation with MightyPPL

The tool MightyPPL [6] was recently developed to generate TBAs from MTL with Past and Pnueli modalities (MTLPPL). The prototype implementation of MoniTAal that we present here requires two complementary automata as input, in place of a temporal logic formula. This is general, in that MoniTAal is capable of monitoring both safety and liveness properties. MTLPPL can, with both future and past modalities, similarly also express more than just safety properties. Note that monitoring a liveness property will never yield the verdict $\bot$, but it is still possible to yield the $\top$ verdict. The property "$F\ a$" is a trivial example of this.

Because MightyPPL translates automata to the automata format used in MoniTAal we can translate MTLPPL formulas with MightyPPL and monitor them with MoniTAal. This allows us to benchmark properties from Timescales [7] and compare with the monitoring tools MonPoly [8] and Reelay [9].

Timescales is a benchmark generator for real-time monitoring tools. It can generate logs (timed words) for 10 safety properties. All of the properties have one upper bounded timing parameter, and some also have a lower bound timing parameter. For these experiments we monitor each property with 3 different sets of parameters. The lower bounds 3, 30, and 300, and the upper bounds 10, 100, and 1000. The following is a list of the Timescales properties with l and u as the lower and upper bound parameter.

```
$ ./MoniTAal-bin <...>

Positive reach-set:
q1 : 0 = 0, x = 0, global = 0

Negative reach-set:
q1 : 0 = 0, x = 0, global = 0

Next event: @20 a
Monitoring ended, verdict is: POSITIVE
Monitored 1 events
--------------------------------
$ ./MoniTAal-bin <...> --div a

Monitoring ended, verdict is: POSITIVE
Monitored 0 events
```

Fig. 12 Example of monitoring property from Example 2 ($F_{\geq 20}a$) with and without time divergence

$$
\begin{aligned}
\text{AbsentAQ: } & G(q \rightarrow G_{[0,u]}\neg p)\\
\text{AbsentBR: } & G(F_{[0,u]}r \rightarrow (\neg p\ U\ r))\\
\text{AbsentBQR: } & G((q \wedge \neg r \wedge F\ r) \rightarrow (\neg p\ U_{[l,u]}\ r))\\
\text{AlwaysAQ: } & G(q \rightarrow G_{[0,u]}\ p)\\
\text{AlwaysBR: } & G(F_{0,u}(r) \rightarrow (p\ U\ r))\\
\text{AlwaysBQR: } & G((q \wedge \neg r \wedge F\ r) \rightarrow (p\ U_{[l,u]}\ r))\\
\text{RecurGLB: } & G(F_{[0,u]}p)\\
\text{RecurBQR: } & G((q \wedge \neg r \wedge F\ r) \rightarrow (F_{[0,u]}(p \vee r)\ U\ r))\\
\text{RespondGLB: } & G(p \rightarrow F_{[l,u]}\ s)\\
\text{RespondBQR: } & G((q \wedge \neg r \wedge F\ r) \rightarrow (p \rightarrow (\neg r\ U_{[l,u]}(\neg r \wedge s))\ U\ r)
\end{aligned}
$$

For our experiments, we used Timescales to generate logs with a time horizon of 1 million and with an observation on every integer unit of time. The logs generated by Timescales are not exactly 1 million events long for every property. but up to 500 events more in some cases. We report on the total running time of the monitor and the peak allocated memory. For MoniTAal we also report the time it took to generate the property automata with MightyPPL. We ran each tool on a Linux (Ubuntu) system with an AMD Opteron 6376 CPU. We recorded user time and maximum resident size using the GNU time program[5]. The automata generation time was recorded internally with the C++ Chrono library.

Although we compare MoniTAal, MonPoly, and Reelay here, these tools have fundamental differences that make a direct comparison less straightforward than it might seem.

[5] https://www.gnu.org/software/time/.

Both Reelay and MonPoly monitor only safety properties using a finite-word semantics. Reelay monitors dense-time past-only Metric Temporal Logic [9] and MonPoly monitors MFOTL using discrete-time semantics [20]. MoniTAal with MightyPPL monitors MTLPPL (MTL with Pnueli, past, and future modalities) and utilizes an infinite word semantics. MightyPPL also implements a slightly different semantics for the Until operator that requires us to extend the formula used by MoniTAal to a slightly larger one than that used by the other tools. As such, a direct comparison of their execution times should be taken with a grain of salt.

Caveats aside, the results in Table 3 show that Reelay is the fastest in all cases, while MonPoly is the slowest in all cases. The table has a row for each property and a column for the running time (in seconds) of each tool. For MonPoly and Reelay we have no measurement of the initial construction time, but for MoniTAal we report the time (in milliseconds) that MightyPPL spent on generating the automaton for the property and its negation in the

Table 3 Results for running MoniTAal (utilizing MightyPPL), MonPoly, and Reelay on properties and data from Timescales. For each tool we report the time (in seconds) it took to monitor a word with 1 million events. For MoniTAal we also report on the time it took to generate the property automata using MightyPPL (in milliseconds). All results are the mean values of 10 runs.

Property	Automata	MoniTAal	MonPoly	Reelay
AbsentAQ10	2± 0	1.47 ± 0.01	5.32 ± 0.06	0.45 ± 0.01
AbsentAQ100	2± 0	1.38 ± 0.01	5.26 ± 0.08	0.43 ± 0.01
AbsentAQ1000	2± 0	1.38 ± 0.02	5.33 ± 0.06	0.45 ± 0.02
AbsentBR10	2± 0	1.67 ± 0.01	4.94 ± 0.06	0.54 ± 0.01
AbsentBR100	2± 0	1.64 ± 0.02	4.94 ± 0.07	0.53 ± 0.01
AbsentBR1000	2± 0	1.64 ± 0.02	4.95 ± 0.07	0.53 ± 0.01
AbsentBQR10	556 ± 5	2.96 ± 0.02	5.72 ± 0.07	0.45 ± 0.01
AbsentBQR100	552 ± 10	2.22 ± 0.02	5.57 ± 0.10	0.43 ± 0.01
AbsentBQR1000	556 ± 1	2.18 ± 0.03	5.53 ± 0.08	0.43 ± 0.01
AlwaysAQ10	2± 0	2.11 ± 0.05	5.65 ± 0.08	0.44 ± 0.01
AlwaysAQ100	2± 0	2.04 ± 0.04	5.66 ± 0.07	0.42 ± 0.01
AlwaysAQ1000	2± 0	2.03 ± 0.02	5.64 ± 0.07	0.42 ± 0.01
AlwaysBR10	4± 2	2.85 ± 0.04	5.74 ± 0.07	0.55 ± 0.01
AlwaysBR100	3± 1	2.83 ± 0.02	5.73 ± 0.04	0.54 ± 0.01
AlwaysBR1000	3± 1	2.81 ± 0.02	5.82 ± 0.05	0.54 ± 0.01
AlwaysBQR10	552 ± 10	3.94 ± 0.02	6.34 ± 0.08	0.46 ± 0.01
AlwaysBQR100	555 ± 1	3.80 ± 0.03	6.36 ± 0.09	0.41 ± 0.01
AlwaysBQR1000	555 ± 4	3.68 ± 0.01	6.34 ± 0.10	0.40 ± 0.02
RecurGLB10	2± 0	1.27 ± 0.01	1.54 ± 0.03	0.62 ± 0.01
RecurGLB100	2± 0	1.11 ± 0.02	1.43 ± 0.02	0.56 ± 0.01
RecurGLB1000	2± 0	1.09 ± 0.02	1.43 ± 0.02	0.55 ± 0.02
RecurBQR10	4± 0	1.39 ± 0.01	5.96 ± 0.10	0.38 ± 0.01
RecurBQR100	4± 0	1.12 ± 0.01	5.89 ± 0.05	0.33 ± 0.01
RecurBQR1000	4± 1	1.10 ± 0.01	5.84 ± 0.08	0.32 ± 0.01
RespondGLB10	559 ± 1	4.91 ± 0.01	5.47 ± 0.07	0.87 ± 0.01
RespondGLB100	555 ± 10	3.24 ± 0.01	5.33 ± 0.11	0.79 ± 0.01
RespondGLB1000	559 ± 1	3.02 ± 0.02	5.30 ± 0.08	0.77 ± 0.01
RespondBQR10	722 ± 10	3.48 ± 0.02	6.54 ± 0.11	0.76 ± 0.01
RespondBQR100	725 ± 1	2.93 ± 0.02	6.38 ± 0.08	0.69 ± 0.01
RespondBQR1000	726 ± 1	2.87 ± 0.02	6.32 ± 0.09	0.67 ± 0.01

Table 4 Results for memory usage in MB from running MoniTAal (utilizing MightyPPL), MonPoly, and Reelay on properties and data from Timescales. All results are the mean values of 10 runs.

Property	MoniTAal	MonPoly	Reelay
bsentAQ10	4 ± 0	4 ± 2	56 ± 2
AbsentAQ100	4 ± 0	4 ± 2	56 ± 2
AbsentAQ1000	4 ± 0	4 ± 2	56 ± 0
AbsentBR10	4 ± 1	4 ± 2	56 ± 2
AbsentBR100	4 ± 0	4 ± 2	56 ± 2
AbsentBR1000	4 ± 2	4 ± 2	56 ± 2
AbsentBQR10	12 ± 3	4 ± 2	68 ± 2
AbsentBQR100	12 ± 1	4 ± 2	69 ± 2
AbsentBQR1000	12 ± 1	4 ± 2	69 ± 0
AlwaysAQ10	4 ± 2	4 ± 2	56 ± 2
AlwaysAQ100	4 ± 2	4 ± 2	56 ± 2
AlwaysAQ1000	4 ± 2	4 ± 2	56 ± 2
AlwaysBR10	4 ± 0	5 ± 2	56 ± 2
AlwaysBR100	4 ± 1	5 ± 2	56 ± 2
AlwaysBR1000	4 ± 0	5 ± 2	56 ± 1
AlwaysBQR10	12 ± 2	4 ± 2	68 ± 2
AlwaysBQR100	12 ± 2	4 ± 2	68 ± 2
AlwaysBQR1000	12 ± 2	4 ± 2	68 ± 2
RecurGLB10	4 ± 2	5 ± 2	45 ± 1
RecurGLB100	4 ± 2	5 ± 2	45 ± 2
RecurGLB1000	4 ± 1	5 ± 2	45 ± 2
RecurBQR10	4 ± 2	4 ± 2	68 ± 1
RecurBQR100	4 ± 2	4 ± 2	69 ± 2
RecurBQR1000	4 ± 2	4 ± 2	69 ± 2
RespondGLB10	12 ± 2	4 ± 2	57 ± 1
RespondGLB100	12 ± 2	4 ± 2	57 ± 2
RespondGLB1000	12 ± 2	4 ± 2	57 ± 2
RespondBQR10	22 ± 2	4 ± 2	80 ± 2
RespondBQR100	22 ± 0	4 ± 2	80 ± 2
RespondBQR1000	22 ± 2	4 ± 2	80 ± 2

"Automata" column. The running time shown for MoniTAal includes both the time spent on monitoring, as well as automata generation. For example, the last row shows results for monitoring the property RespondBQR with a lower bound of 300 and upper bound of 1000. For this property (RespondBQR1000), MoniTAal spent 726 milliseconds on initial automata generation and ran for 2.87 seconds in total while MonPoly spent 6.32 seconds and Reelay spent 0.67 seconds.

Table 4 shows the maximum allocated memory for each monitor in MB. The table has a row for each property as well as a column for each tool showing their memory usage for each property. We see that Reelay uses the most memory in all cases, ranging between 56 and 80 MB, while MonPoly is consistently around 4 MB and MoniTAal is between 4 and 22 MB. Note that this might not fully represent the memory usage in practical applications of these tools. For example, each monitor is provided a log of data representing a timed word. The required format of this log affects the size of the log file, which might affect the

memory usage of the monitor. Reelay's log format is more verbose than the MonPoly format used by the other tools.

It is clear from empirical observation that the memory usage of MoniTAal is closely related to the size of the generated automata. AbsentBQR, AlwaysBQR, RespondGLB, and RespondBQR were the only properties for which it took longer than 4 ms to generate the property automata. These are also the only properties for which MoniTAal used more than 4 MB of memory. The reason is that the automata for these properties were much larger than the rest.

9 Conclusion

In this work, we have studied 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. In particular, we showed how to account for time divergence which prior work did not readily seem to support.

We also considered an extension of our method supporting timing uncertainty in the time sequence observed by the monitor. By supporting timing uncertainty 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.

The monitoring algorithms for both concrete and uncertain traces have been implemented in the tool MoniTAal. The tool facilitates intersecting TBAs (also with the time divergence automaton), computing the non-empty language states, reading timed words, and online monitoring. We aim to exploit the symbolic monitoring engine of MoniTAal to replace the current rewrite-based Weighted MTL implementation [21] of Uppaal SMC [52], as this only supports a fragment of MITL.

Further improvements to our monitoring algorithm include improved support for uncertainties in the input and additional analysis of timed properties. In recent work, we introduced support for unobservable symbols in the monitor alphabet [34]. This can be logically extended to consider arbitrary mutations to an input sequence modeled in the form of a TA.

Acknowledgements We would like to express our sincere gratitude to Hsi-Ming Ho for helping us integrating the MightyPPL tool. T.M. Grosen, K.G. Larsen, and M. Zimmermann have been supported by DIREC - Digital Research Centre Denmark. T.M. Grosen has been supported by Mitacs. S. Kauffman has been supported by NSERC - Natural Sciences and Engineering Research Council of Canada.

Author contributions TMG implemented the tool MoniTAal and ran all experiments. All authors wrote and reviewed the manuscript.

Funding Open access funding provided by Aalborg University

Data availability No datasets were generated or analysed during the current study.

Declarations

Supplementary information The tool implementing the algorithms presented here is freely available at https://github.com/DEIS-Tools/MoniTAal.

Competing interests The authors declare no competing interests.

References

1. Koymans R (1990) Specifying real-time properties with metric temporal logic. Real-Time Syst 2(4):255–299. https://doi.org/10.1007/BF01995674
2. Alur R, Feder T, Henzinger, Henzinger TA (1996) The benefits of relaxing punctuality. JACM 43(1):116–146. https://doi.org/10.1145/227595.227602
3. Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235. https://doi.org/10.1016/0304-3975(94)90010-8
4. Brihaye T, Geeraerts G, Ho H, Monmege B (2017) MightyL: a compositional translation from MITL to timed automata. In: Majumdar R, Kuncak V (eds) CAV 2017, part I. LNCS. Springer, pp 421–440. https://doi.org/10.1007/978-3-319-63387-9_21
5. Bengtsson J, Yi W (2003) Timed automata: semantics, algorithms and tools. In: Desel J, Reisig W, Rozenberg G (eds) Lectures on concurrency and Petri nets, advances in Petri nets. LNCS, vol 3098. Springer, Cham, pp 87–124. https://doi.org/10.1007/978-3-540-27755-2_3
6. Ho H, Krishna SN, Madnani K, Majumdar R, Pandya PK (2026) MightyPPL: model checking MITL with past and Pnueli modalities. In: Junges S, Katz G (eds) TACAS 2026, part I. LNCS. Springer, pp 457–479. https://doi.org/10.1007/978-3-032-22752-2_24
7. Ulus D (2019) Timescales: a benchmark generator for MTL monitoring tools. In: Finkbeiner B, Mariani L (eds) RV 2019. LNCS. Springer, pp 402–412. https://doi.org/10.1007/978-3-030-32079-9_25
8. Basin DA, Harvan M, Klaedtke F, Zalinescu E (2011) MONPOLY: monitoring usage-control policies. In: Khurshid S, Sen K (eds) RV 2011. LNCS. Springer, pp 360–364. https://doi.org/10.1007/978-3-642-29860-8_27
9. Ulus D (2026) Online monitoring of metric temporal logic using sequential networks. Log Meth Comput Sci 22(1). https://doi.org/10.46298/LMCS-22(1:12)2026
10. Grosen TM, Kauffman S, Larsen KG, Zimmermann M (2022) Monitoring timed properties (revisited). In: Bogomolov S, Parker D (eds) FORMATS 2022. LNCS, vol 13465. Springer, Cham, pp 43–62. https://doi.org/10.1007/978-3-031-15839-1_3
11. Thati P, Rosu G (2004) Monitoring algorithms for metric temporal logic specifications. In: Havelund K, Rosu G (eds) RV 2004. ENTCS, vol 113. Elsevier, pp 145–162. https://doi.org/10.1016/J.ENTCS.2004.01.029
12. Basin DA, Klaedtke F, Zalinescu E (2018) Algorithms for monitoring real-time properties. Acta Inf 55(4):309–338. https://doi.org/10.1007/S00236-017-0295-4
13. Ulus D, Ferrère T, Asarin E, Maler O (2014) Timed pattern matching. In: Legay A, Bozga M (eds) FORMATS 2014. LNCS, vol 8711. Springer, Cham, pp 222–236. https://doi.org/10.1007/978-3-319-10512-3_16
14. Ulus D, Ferrère T, Asarin E, Maler O (2016) Online timed pattern matching using derivatives. In: Chechik M, Raskin J (eds) TACAS 2016. LNCS, vol 9636. Springer, Cham, pp 736–751. https://doi.org/10.1007/978-3-662-49674-9_47

15. Bauer A, Leucker M, Schallhart C (2006) Monitoring of real-time properties. In: Arun-Kumar S, Garg N (eds) FSTTCS 2006. LNCS, vol 4337. Springer, Cham, pp 260–272. https://doi.org/10.1007/11944836_25
16. Yannakakis M, Lee D (1997) An efficient algorithm for minimizing real-time transition systems. Formal Methods System Des 11(2):113–136. https://doi.org/10.1023/A:1008621829508
17. Larsen KG, Pettersson P, Yi W (1995) Model-checking for real-time systems. In: Reichel H (ed) FCT 1995. LNCS, vol 965. Springer, pp 62–88. https://doi.org/10.1007/3-540-60249-6_41
18. Baldor K, Niu J (2012) Monitoring dense-time, continuous-semantics, metric temporal logic. In: Qadeer S, Tasiran S (eds) RV 2012. LNCS, vol 7687. Springer, Cham, pp 245–259. https://doi.org/10.1007/978-3-642-35632-2_24
19. Ho H, Ouaknine J, Worrell J (2014) Online monitoring of metric temporal logic. In: Bonakdarpour B, Smolka SA (eds) RV 2014. LNCS, vol 8734. Springer, Cham, pp 178–192. https://doi.org/10.1007/978-3-319-11164-3_15
20. Basin DA, Klaedtke F, Müller S, Pfitzmann B (2008) Runtime monitoring of metric first-order temporal properties. In: Hariharan R, Mukund M, Vinay V (eds) FSTTCS 2008. LIPIcs, vol 2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp 49–60. https://doi.org/10.4230/LIPICS.FSTTCS.2008.1740
21. Bulychev PE, David A, Larsen KG, Legay A, Li G, Poulsen DB (2012) Rewrite-based statistical model checking of WMTL. In: Qadeer S, Tasiran S (eds) RV 2012. LNCS, vol 7687. Springer, Cham, pp 260–275. https://doi.org/10.1007/978-3-642-35632-2_25
22. Moosbrugger P, Rozier KY, Schumann J (2017) R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. Form Methods Syst Des 51(1):31–61. https://doi.org/10.1007/S10703-017-0275-X
23. Chattopadhyay A, Mamouras K (2020) A verified online monitor for metric temporal logic with quantitative semantics. In: Deshmukh J, Nickovic D (eds) RV 2020. LNCS, vol 12399. Springer, Cham, pp 383–403. https://doi.org/10.1007/978-3-030-60508-7_21
24. Basin DA, Krstic S, Traytel D (2017) AERIAL: almost event-rate independent algorithms for monitoring metric regular properties. In: Reger G, Havelund K (eds) RV-CuBES 2017. Kalpa publications in computing. EasyChair, pp 29–36. https://doi.org/10.29007/BM4C
25. Ulus D (2017) Montre: a tool for monitoring timed regular expressions. In: Majumdar R, Kuncak V (eds) CAV 2017, part I. LNCS. Springer, pp 329–335. https://doi.org/10.1007/978-3-319-63387-9_16
26. Nickovic D, Maler O (2007) AMT: a property-based monitoring tool for analog systems. In: Raskin J, Thiagarajan PS (eds) FORMATS 2007. LNCS. Springer, pp 304–319. https://doi.org/10.1007/978-3-540-75454-1_22
27. Nickovic D, Yamaguchi T (2020) RTAMT: online robustness monitors from STL. In: Hung DV, Sokolsky O (eds) ATVA 2020. LNCS. Springer, pp 564–571. https://doi.org/10.1007/978-3-030-59152-6_34
28. Henzinger TA, Manna Z, Pnueli A (1992) What good are digital clocks? In: Kuich W (ed) ICALP92. LNCS. Springer, pp 545–558. https://doi.org/10.1007/3-540-55719-9_103
29. Havelund K, Peled D (2020) First-order timed runtime verification using BDDs. In: Hung DV, Sokolsky O (eds) ATVA 2020. LNCS. Springer, pp 3–24. https://doi.org/10.1007/978-3-030-59152-6_1
30. Henry L, Jéron T, Markey N, Roussanaly V (2024) Distributed monitoring of timed properties. In: Ábrahám E, Abbas H (eds) RV 2024. LNCS. Springer, pp 243–261. https://doi.org/10.1007/978-3-031-74234-7_16
31. Grosen TM, Kauffman S, Larsen KG, Zimmermann M (2025) Time for timed monitorability. In: Bouyer P, Pol J (eds) CONCUR 2025. LIPIcs. pp 19–11920. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (https://doi.org/10.4230/LIPICS.CONCUR.2025.19
32. Amara M, Bernardi G, Foughali MA, Francalanza A (2025) A theory of (linear-time) timed monitors. In: Aldrich J, Silva A (eds) ECOOP 2025. LIPIcs. pp 1–1130. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (https://doi.org/10.4230/LIPICS.ECOOP.2025.1
33. Fränzle M, Grosen TM, Larsen KG, Zimmermann M (2024) Monitoring real-time systems under parametric delay. In: Kosmatov N, Kovács L (eds) IFM 2024. LNCS, vol 15234. Springer, pp 194–213. https://doi.org/10.1007/978-3-031-76554-4_11
34. Cimatti A, Grosen TM, Larsen KG, Tonetta S, Zimmermann M (2024) Exploiting assumptions for effective monitoring of real-time properties under partial observability. In: Madeira A, Knapp A (eds) SEFM 2024. LNCS, vol 15280. Springer, pp 70–88. https://doi.org/10.1007/978-3-031-77382-2_5
35. Li G, Jensen PG, Larsen KG, Legay A, Poulsen DB (2017) Practical controller synthesis for MTL_{∞}. In: Erdogmus H, Havelund K (eds) SPIN 2017. ACM, New York, NY, USA, pp 102–111. https://doi.org/10.1145/3092282.3092303
36. Bulychev PE, David A, Larsen KG, Legay A, Li G, Poulsen DB, Stainer A (2012) Monitor-based statistical model checking for weighted metric temporal logic. In: Bjørner NS, Voronkov A (eds) LPAR 2018. LNCS, vol 7180. Springer, Cham, pp 168–182. https://doi.org/10.1007/978-3-642-28717-6_15

37. Geilen M, Dams D (2000) An on-the-fly tableau construction for a real-time temporal logic. In: Joseph M (ed) FTRTFT 2000. LNCS, vol 1926. Springer, Cham, pp 276–290. https://doi.org/10.1007/3-540-45352-0_23
38. Finkbeiner B, Kuhtz L (2009) Monitor circuits for LTL with bounded and unbounded future. In: Bensalem S, Peled DA (eds) RV 2009. LNCS, vol 5779. Springer, Cham, pp 60–75. https://doi.org/10.1007/978-3-642-04694-0_5
39. Nickovic D, Piterman N (2010) From MTL to deterministic timed automata. In: Chatterjee K, Henzinger TA (eds) FORMATS 2010. LNCS, vol 6246. Springer, Cham, pp 152–167. https://doi.org/10.1007/978-3-642-15297-9_13
40. Clarke EM, Henzinger TA, Veith H, Bloem R (eds.) (2018) Handbook of Model Checking. Springer. https://doi.org/10.1007/978-3-319-10575-8
41. Lindahl M, Pettersson P, Yi W (2001) Formal design and analysis of a gear controller. STTT 3(3):353–368. https://doi.org/10.1007/S100090100048
42. Pnueli A, Zaks A (2006) PSL model checking and run-time verification via testers. In: Misra J, Nipkow T, Sekerinski E (eds) FM 2006. LNCS, vol 4085. Springer, Cham, pp 573–586. https://doi.org/10.1007/11813040_38
43. Bauer A, Leucker M, Schallhart C (2011) Runtime verification for LTL and TLTL. ACM Trans Softw Eng Methodol 20(4):1–64. https://doi.org/10.1145/2000799.2000800
44. Kauffman S, Havelund K, Fischmeister S (2021) What can we monitor over unreliable channels? Int J Softw Tools Technol Transf 23(4):579–600. https://doi.org/10.1007/S10009-021-00625-Z
45. Bouyer P, Laroussinie F (2008) 4. Model checking timed automata. John Wiley & Sons, Ltd, pp 111–140. https://doi.org/10.1002/9780470611012.ch4
46. Jahanian F, Rajkumar R, Raju SCV (1994) Runtime monitoring of timing constraints in distributed real-time systems. Real-Time Syst 7(3):247–273. https://doi.org/10.1007/BF01088521
47. Rushby JM (1999) Systematic formal verification for fault-tolerant time-triggered algorithms. IIEEE Trans. Softw Eng 25(5):651–660. https://doi.org/10.1109/32.815324
48. Pike L (2006) A note on inconsistent axioms in Rushby's "systematic formal verification for fault-tolerant time-triggered algorithms". IIEEE Trans. Softw Eng 32(5):347–348. https://doi.org/10.1109/TSE.2006.41
49. Wulf MD, Doyen L, Markey N, Raskin J (2008) Robust safety of timed automata. Formal Methods Syst. Form Methods Syst Des 33(1–3):45–84. https://doi.org/10.1007/S10703-008-0056-7
50. Kauffman S, Havelund K, Fischmeister S (2019) Monitorability over unreliable channels. In: Finkbeiner B, Mariani L (eds) RV 2019. LNCS, vol 11757. Springer, Cham, pp 256–272. https://doi.org/10.1007/978-3-030-32079-9_15
51. Daws C, Yovine S (1996) Reducing the number of clock variables of timed automata. RTSS. IEEE Computer Society, In, pp 73–81. https://doi.org/10.1109/REAL.1996.563702
52. David A, Larsen KG, Legay A, Mikucionis M, Wang Z (2011) Time for statistical model checking of real-time systems. In: Gopalakrishnan G, Qadeer S (eds) CAV 2011. LNCS, vol 6806. Springer, Cham, pp 349–355. https://doi.org/10.1007/978-3-642-22110-1_27