Algorithms for Monitoring Real-Time Properties[⋆,⋆⋆]

David Basin, Felix Klaedtke, and Eugen Zălinescu

Computer Science Department, ETH Zurich, Switzerland

Abstract. We present and analyze monitoring algorithms for a safety fragment of metric temporal logics, which differ in their underlying time model. The time models considered have either dense or discrete time domains and are point-based or interval-based. Our analysis reveals differences and similarities between the time models for monitoring and highlights key concepts underlying our and prior monitoring algorithms.

1 Introduction

Real-time logics [2] allow us to specify system properties involving timing constraints, e.g., every request must be followed within 10 seconds by a grant. Such specifications are useful when designing, developing, and verifying systems with hard real-time requirements. They also have applications in runtime verification, where monitors generated from specifications are used to check the correctness of system behavior at runtime [10]. Various monitoring algorithms for real-time logics have been developed [4, 5, 7, 12, 14, 15, 17, 20] based on different time models. These time models can be characterized by two independent aspects. First, a time model is either point-based or interval-based. In point-based time models, system traces are sequences of system states, where each state is time-stamped. In interval-based time models, system traces consist of continuous (Boolean) signals of state variables. Second, a time model is either dense or discrete depending on the underlying ordering on time-points, i.e., whether there are infinitely many or finitely many time-points between any two distinct time-points.

Real-time logics based on a dense, interval-based time model are more natural and general than their counterparts based on a discrete or point-based model. In fact, both discrete and point-based time models can be seen as abstractions of dense, interval-based time models [2, 18]. However, the satisfiability and the model-checking problems for many real-time logics with the more natural time model are computationally harder than their corresponding decision problems when the time model is discrete or point-based. See the survey [16] for further discussion and examples.

In this paper, we analyze the impact of different time models on monitoring. We do this by presenting, analyzing, and comparing monitoring algorithms

[⋆] This work was supported by the Nokia Research Center, Switzerland.

[⋆⋆] Due to space restrictions, some proof details have been omitted. They can be found in the full version of the paper, which is available from the authors' web pages.

© Springer-Verlag Berlin Heidelberg 2012

for real-time logics based on different time models. More concretely, we present monitoring algorithms for the past-only fragment of propositional metric temporal logics with a point-based and an interval-based semantics, also considering both dense and discrete time domains. We compare our algorithms on a class of formulas for which the point-based and the interval-based settings coincide. To define this class, we distinguish between event propositions and state propositions. The truth value of a state proposition always has a duration, whereas an event proposition cannot be continuously true between two distinct time-points.

Our analysis explains the impact of different time models on monitoring. First, the impact of a dense versus a discrete time domain is minor. The algorithms are essentially the same and have almost identical computational complexities. Second, monitoring in a point-based setting is simpler than in an interval-based setting. The meaning of "simpler" is admittedly informal here since we do not provide lower bounds. However, we consider our monitoring algorithms for the point-based setting as conceptually simpler than the interval-based algorithms. Moreover, we show that our point-based monitoring algorithms perform better than our interval-based algorithms on the given class of formulas on which the two settings coincide.

Overall, we see the contributions as follows. First, our monitoring algorithms simplify and clarify key concepts of previously presented algorithms [4, 13–15]. In particular, we present the complete algorithms along with a detailed complexity analysis for monitoring properties specified in the past-only fragment of propositional metric temporal logic. Second, our monitoring algorithm for the dense, point-based time model has better complexity bounds than existing algorithms for the same time model [20]. Third, our comparison of the monitoring algorithms illustrates the similarities, differences, and trade-offs between the time models with respect to monitoring. Moreover, formulas in our fragment benefit from both settings: although they describe properties based on a more natural time model, they can be monitored with respect to a point-based time model, which is more efficient.

2 Preliminaries

Time Domain and Intervals. If not stated differently, we assume the dense time domain[1] $\mathbb{T} = \mathbb{Q}_{\geq 0}$ with the standard ordering $\leq$. Adapting the following definitions to a discrete time domain like $\mathbb{N}$ is straightforward.

A *(time) interval* is a non-empty set $I \subseteq \mathbb{T}$ such that if $\tau < \kappa < \tau'$ then $\kappa \in I$, for all $\tau, \tau' \in I$ and $\kappa \in \mathbb{T}$. We denote the set of all time intervals by $\mathbb{I}$. An interval is either left-open or left-closed and similarly either right-open or right-closed. We denote the left margin and the right margin of an interval $I \in \mathbb{I}$ by $\ell(I)$ and

[1] We do not use $\mathbb{R}_{\geq 0}$ as dense time domain because of representation issues. Namely, each element in $\mathbb{Q}_{\geq 0}$ can be finitely represented, which is not the case for $\mathbb{R}_{\geq 0}$. Choosing $\mathbb{Q}_{\geq 0}$ instead of $\mathbb{R}_{\geq 0}$ is without loss of generality for the satisfiability of properties specified in real-time logics like the metric interval temporal logic [1].

$r(I)$, respectively. For instance, the interval $I = \{\tau \in \mathbb{T} \mid 3 \leq \tau\}$, which we also write as $[3, \infty)$, is left-closed and right-open with margins $\ell(I) = 3$ and $r(I) = \infty$.

For an interval $I \in \mathbb{I}$, we define the extension $I^{\geq} := I \cup (\ell(I), \infty)$ to the right and its strict counterpart $I^{>} := I^{\geq} \setminus I$, which excludes I. We define ${}^{\leq}I := [0, r(I)) \cup I$ and ${}^{<}I := ({}^{\leq}I) \setminus I$ similarly. An interval $I \in \mathbb{I}$ is *singular* if $|I| = 1$, *bounded* if $r(I) < \infty$, and *unbounded* if $r(I) = \infty$. The intervals $I, J \in \mathbb{I}$ are *adjacent* if $I \cap J = \emptyset$ and $I \cup J \in \mathbb{I}$. For $I, J \in \mathbb{I}$, $I \oplus J$ is the set $\{\tau + \tau' \mid \tau \in I \text{ and } \tau' \in J\}$.

An *interval partition* of $\mathbb{T}$ is a sequence $\langle I_i \rangle_{i \in N}$ of time intervals with $N = \mathbb{N}$ or $N = \{0, \ldots, n\}$ for some $n \in \mathbb{N}$ that fulfills the following properties: (i) I_{i-1} and I_i are adjacent and $\ell(I_{i-1}) \leq \ell(I_i)$, for all $i \in N \setminus \{0\}$, and (ii) for each $\tau \in \mathbb{T}$, there is an $i \in N$ such that $\tau \in I_i$. The interval partition $\langle J_j \rangle_{j \in M}$ *refines* the interval partition $\langle I_i \rangle_{i \in N}$ if for every $j \in M$, there is some $i \in N$ such that $J_j \subseteq I_i$. We often write $\bar{I}$ for a sequence of intervals instead of $\langle I_i \rangle_{i \in N}$. Moreover, we abuse notation by writing $I \in \langle I_i \rangle_{i \in N}$ if $I = I_i$, for some $i \in N$.

A *time sequence* $\langle \tau_i \rangle_{i \in \mathbb{N}}$ is a sequence of elements $\tau_i \in \mathbb{T}$ that is strictly increasing (i.e., $\tau_i < \tau_j$, for all $i, j \in \mathbb{N}$ with $i < j$) and progressing (i.e., for all $\tau \in \mathbb{T}$, there is $i \in \mathbb{N}$ with $\tau_i > \tau$). Similar to interval sequences, $\bar{\tau}$ abbreviates $\langle \tau_i \rangle_{i \in \mathbb{N}}$.

Boolean Signals. A *(Boolean) signal* γ is a subset of $\mathbb{T}$ that fulfills the following finite-variability condition: for every bounded interval $I \in \mathbb{I}$, there are intervals $I_0, \ldots, I_{n-1} \in \mathbb{I}$ such that $\gamma \cap I = I_0 \cup \cdots \cup I_{n-1}$, for some $n \in \mathbb{N}$. The least such $n \in \mathbb{N}$ is the *size* of the signal γ on I. We denote it by $\|\gamma \cap I\|$.

We use the term "signal" for such a set γ because its characteristic function $\chi_\gamma : \mathbb{T} \to \{0, 1\}$ represents, for example, the values over time of an input or an output of a sequential circuit. Intuitively, $\tau \in \gamma$ iff the signal of the circuit is high at the time $\tau \in \mathbb{T}$. The finite-variability condition imposed on the set γ prevents switching infinitely often from high to low in finite time. Note that $\|\gamma \cap I\|$ formalizes how often a signal γ is high on the bounded interval I, in particular, $\|\gamma \cap I\| = 0$ iff $\gamma \cap I = \emptyset$.

A signal γ is *stable* on an interval $I \in \mathbb{I}$ if $I \subseteq \gamma$ or $I \cap \gamma = \emptyset$. The *induced interval partition* $\overline{\mathsf{iip}}(\gamma)$ of a signal γ is the interval partition $\bar{I}$ such that γ is stable on each of the intervals in $\bar{I}$ and any other stable interval partition refines $\bar{I}$. We write $\overline{\mathsf{iip}}^1(\gamma)$ for the sequence of intervals I in $\overline{\mathsf{iip}}(\gamma)$ such that $I \cap \gamma \neq \emptyset$. Similarly, we write $\overline{\mathsf{iip}}^0(\gamma)$ for the sequence of intervals I in $\overline{\mathsf{iip}}(\gamma)$ such that $I \cap \gamma = \emptyset$. Intuitively, $\overline{\mathsf{iip}}^1(\gamma)$ and $\overline{\mathsf{iip}}^0(\gamma)$ are the sequences of maximal intervals on which the signal is γ is high and low, respectively.

Metric Temporal Logics. To simplify the exposition, we restrict ourselves to monitoring the past-only fragment of metric temporal logic in a point-based and an interval-based setting. However, future operators like $\Diamond_I$, where the interval I is bounded, can be handled during monitoring by using queues that postpone the evaluation until enough time has elapsed. See [4], for such a monitoring algorithm that handles arbitrary nesting of past and bounded future operators.

Let P be a non-empty set of *propositions*. The syntax of the past-only fragment of metric temporal logic is given by the grammar $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathsf{S}_I \phi$, where $p \in P$ and $I \in \mathbb{I}$. In Figure 1, we define the satisfaction relations $\models$ and $\mathrel{\dot{\models}}$,

$$\begin{array}{ll}
\hat{\gamma}, \tau \models p & \text{iff } \tau \in \gamma_p \\
\hat{\gamma}, \tau \models \neg\phi & \text{iff } \hat{\gamma}, \tau \not\models \phi \\
\hat{\gamma}, \tau \models \phi \wedge \psi & \text{iff } \hat{\gamma}, \tau \models \phi \text{ and } \hat{\gamma}, \tau \models \psi \\
\hat{\gamma}, \tau \models \phi \mathsf{S}_I \psi & \text{iff there is } \tau' \in [0, \tau] \text{ with} \\
& \quad \tau - \tau' \in I, \\
& \quad \hat{\gamma}, \tau' \models \psi, \text{ and} \\
& \quad \hat{\gamma}, \kappa \models \phi, \text{ for all } \kappa \in (\tau', \tau]
\end{array}$$

(a) interval-based semantics

$$\begin{array}{ll}
\hat{\gamma}, \bar{\tau}, i \dot{\models} p & \text{iff } \tau_i \in \gamma_p \\
\hat{\gamma}, \bar{\tau}, i \dot{\models} \neg\phi & \text{iff } \hat{\gamma}, \bar{\tau}, i \not\dot{\models} \phi \\
\hat{\gamma}, \bar{\tau}, i \dot{\models} \phi \wedge \psi & \text{iff } \hat{\gamma}, \bar{\tau}, i \dot{\models} \phi \text{ and } \hat{\gamma}, \bar{\tau}, i \dot{\models} \psi \\
\hat{\gamma}, \bar{\tau}, i \dot{\models} \phi \mathsf{S}_I \psi & \text{iff there is } i' \in [0, i] \cap \mathbb{N} \text{ with} \\
& \quad \tau_i - \tau_{i'} \in I, \\
& \quad \hat{\gamma}, \bar{\tau}, i' \dot{\models} \psi, \text{ and} \\
& \quad \hat{\gamma}, \bar{\tau}, k \dot{\models} \phi, \text{ for all } k \in (i', i] \cap \mathbb{N}
\end{array}$$

(b) point-based semantics

Fig. 1. Semantics of past-only metric temporal logic

where $\hat{\gamma} = (\gamma_p)_{p \in P}$ is a family of signals, $\bar{\tau}$ a time sequence, $\tau \in \mathbb{T}$, and $i \in \mathbb{N}$. Note that $\models$ defines the truth value of a formula for every $\tau \in \mathbb{T}$. In contrast, a formula's truth value with respect to $\dot{\models}$ is defined at the "sample-points" $i \in \mathbb{N}$ to which the "time-stamps" $\tau_i \in \mathbb{T}$ from the time sequence $\bar{\tau}$ are attached.

We use the standard binding strength of the operators and standard syntactic sugar. For instance, $\phi \vee \psi$ stands for the formula $\neg(\neg\phi \wedge \neg\psi)$ and $\blacklozenge_I \psi$ stands for $(p \vee \neg p) \mathsf{S}_I \psi$, for some $p \in P$. Moreover, we often omit the interval $I = [0, \infty)$ attached to a temporal operator. We denote the set of subformulas of a formula ϕ by $\mathsf{sf}(\phi)$. Finally, $|\phi|$ is the number of nodes in ϕ's parse tree.

3 Point-Based versus Interval-Based Time Models

3.1 State Variables and System Events

State variables and system events are different kinds of entities. One distinguishing feature is that events happen at single points in time and the value of a state variable is always constant for some amount of time. In the following, we distinguish between these two entities. Let P be the disjoint union of the proposition sets S and E. We call propositions in S *state propositions* and propositions in E *event propositions*. Semantically, a signal $\gamma \subseteq \mathbb{T}$ is an *event signal* if $\gamma \cap I$ is finite, for every bounded interval I, and the signal γ is a *state signal* if for every bounded interval I, the sets $\gamma \cap I$ and $(\mathbb{T} \setminus \gamma) \cap I$ are the finite unions of non-singular intervals. Note that there are signals that are neither event signals nor state signals. A family of signals $\hat{\gamma} = (\gamma_p)_{p \in S \cup E}$ is *consistent* with S and E if γ_p is a state signal, for all $p \in S$, and γ_p is an event signal, for all $p \in E$.

The point-based semantics is often motivated by the study of real-time systems whose behavior is determined by system events. Intuitively, a time sequence $\bar{\tau}$ records the points in time when events occur and the signal γ_p for a proposition $p \in E$ consists of the points in time when the event p occurs. The following examples, however, demonstrate that the point-based semantics can be unintuitive in contrast to the interval-based semantics.

Example 1. A state proposition $p \in S$ can often be mimicked by the formula $\neg f \mathsf{S}\, s$ with corresponding event propositions $s, f \in E$ representing "start" and "finish." For the state signal γ_p, let γ_s and γ_f be the event signals where γ_s and γ_f consist of the points in time of γ_p when the Boolean state variable starts and respectively finishes to hold. Then $(\gamma_s, \gamma_f), \tau \models \neg f \mathsf{S}\, s$ iff $\gamma_p, \tau \models p$, for any $\tau \in \mathbb{T}$, under the assumption that $I \cap \gamma_p$ is the finite union of left-closed and right-open intervals, for every bounded left-closed and right-open interval I.

However, replacing p by $\neg f \mathsf{S}\, s$ does not always capture the essence of a Boolean state variable when using the point-based semantics. Consider the formula $\blacklozenge_{[0,1]}\, p$ containing the state proposition p and let $\gamma_p = [0, 5)$ be a state signal. Moreover, let (γ_s, γ_f) be the family of corresponding event signals for the event propositions s and f, i.e., $\gamma_s = \{0\}$ and $\gamma_f = \{5\}$. For a time sequence $\bar\tau$ with $\tau_0 = 0$ and $\tau_1 = 5$, we have that $(\gamma_s, \gamma_f), \bar\tau, 1 \not\models \blacklozenge_{[0,1]}(\neg f \mathsf{S}\, s)$ but $\gamma_p, \tau_1 \models \blacklozenge_{[0,1]}\, p$. Note that $\bar\tau$ only contains time-stamps when an event occurs. An additional sample-point between τ_0 and τ_1 with, e.g., the time-stamp 4 would result in identical truth values at time 5.

Example 2. Consider the (event) signals $\gamma_p = \{\tau \in \mathbb{T} \mid \tau = 2n, \text{ for some } n \in \mathbb{N}\}$ and $\gamma_q = \emptyset$ for the (event) propositions p and q. One might expect that these signals satisfy the formula $p \to \blacklozenge_{[0,1]}\, \neg q$ at every point in time. However, for a time sequence $\bar\tau$ with $\tau_0 = 0$ and $\tau_1 = 2$, we have that $\hat\gamma, \bar\tau, 1 \not\mathrel{\dot\models} p \to \blacklozenge_{[0,1]}\, \neg q$. The reason is that in the point-based semantics, the $\blacklozenge_I$ operator requires the existence of a previous point in time that also occurs in the time sequence $\bar\tau$.

As another example consider the formula $\blacklozenge_{[0,1]} \blacklozenge_{[0,1]}\, p$. One might expect that it is logically equivalent to $\blacklozenge_{[0,2]}\, p$. However, this is not the case in the point-based semantics. To see this, consider a time sequence $\bar\tau$ with $\tau_0 = 0$ and $\tau_1 = 2$. We have that $\hat\gamma, \bar\tau, 1 \not\mathrel{\dot\models} \blacklozenge_{[0,1]} \blacklozenge_{[0,1]}\, p$ and $\hat\gamma, \bar\tau, 1 \mathrel{\dot\models} \blacklozenge_{[0,2]}\, p$ if $\tau_0 \in \gamma_p$.

The examples above suggest that adding additional sample-points restores a formula's intended meaning, which usually stems from having the interval-based semantics in mind. However, a drawback of this approach for monitoring is that each additional sample-point increases the workload of a point-based monitoring algorithm, since it is invoked for each sample-point. Moreover, in the dense time domain, adding sample-points does not always make the two semantics coincide. For instance, for $\gamma_p = [0, 1)$ and $\tau \geq 1$, we have that $\gamma_p, \tau \not\models \neg p \mathsf{S}\, p$ and $\gamma_p, \bar\tau, i \mathrel{\dot\models} \neg p \mathsf{S}\, p$, for every time sequence $\bar\tau$ with $\tau_0 < 1$ and every $i \in \mathbb{N}$.

3.2 Event-Relativized Formulas

In the following, we identify a class of formulas for which the point-based and the interval-based semantics coincide. For formulas in this class, a point-based monitoring algorithm can be used to soundly monitor properties given by formulas interpreted using the interval-based semantics. We assume that the propositions are typed, i.e., $P = S \cup E$, where S contains the state propositions and E the event propositions, and a family of signals $\hat\gamma = (\gamma_p)_{p \in S \cup E}$ is consistent with S and E. Moreover, we assume without loss of generality that there is always at least one event signal γ in $\hat\gamma$ that is the infinite union of singular intervals, e.g., γ is the signal of a clock event that regularly occurs over time.

We inductively define the sets $\mathit{rel}_\forall$ and $\mathit{rel}_\exists$ for formulas in negation normal form. Recall that a formula is in negation normal form if negation only occurs directly in front of propositions. A logically-equivalent negation normal form of a formula can always be obtained by eliminating double negations and by pushing negations inwards, where we consider the Boolean connective $\vee$ and the temporal operator "trigger" T_I as primitives. Note that $\phi \,\mathsf{T}_I\, \psi = \neg(\neg\phi \,\mathsf{S}_I\, \neg\psi)$.

$$\begin{aligned}
\neg p &\in \mathit{rel}_\forall && \text{if} \quad p \in E && (\forall 1)\\
\phi_1 \vee \phi_2 &\in \mathit{rel}_\forall && \text{if} \quad \phi_1 \in \mathit{rel}_\forall \text{ or } \phi_2 \in \mathit{rel}_\forall && (\forall 2)\\
\phi_1 \wedge \phi_2 &\in \mathit{rel}_\forall && \text{if} \quad \phi_1 \in \mathit{rel}_\forall \text{ and } \phi_2 \in \mathit{rel}_\forall && (\forall 3)\\
p &\in \mathit{rel}_\exists && \text{if} \quad p \in E && (\exists 1)\\
\phi_1 \wedge \phi_2 &\in \mathit{rel}_\exists && \text{if} \quad \phi_1 \in \mathit{rel}_\exists \text{ or } \phi_2 \in \mathit{rel}_\exists && (\exists 2)\\
\phi_1 \vee \phi_2 &\in \mathit{rel}_\exists && \text{if} \quad \phi_1 \in \mathit{rel}_\exists \text{ and } \phi_2 \in \mathit{rel}_\exists && (\exists 3)
\end{aligned}$$

A formula ϕ is *event-relativized* if $\alpha \in \mathit{rel}_\forall$ and $\beta \in \mathit{rel}_\exists$, for every subformula of ϕ of the form $\alpha \,\mathsf{S}_I\, \beta$ or $\beta \,\mathsf{T}_I\, \alpha$. We call the formula ϕ *strongly* event-relativized if ϕ is event-relativized and $\phi \in \mathit{rel}_\forall \cup \mathit{rel}_\exists$.

The following theorem relates the interval-based semantics and the point-based semantics for event-relativized formulas.

Theorem 1. *Let $\hat{\gamma} = (\gamma_p)_{p \in S \cup E}$ be a family of consistent signals and $\bar{\tau}$ the time sequence listing the occurrences of events in $\hat{\gamma}$, i.e., $\bar{\tau}$ is the time sequence obtained by linearly ordering the set $\bigcup_{p \in E} \gamma_p$. For an event-relativized formula ϕ and every $i \in \mathbb{N}$, it holds that $\hat{\gamma}, \tau_i \models \phi$ iff $\hat{\gamma}, \bar{\tau}, i \mathrel{\dot{\models}} \phi$. Furthermore, if ϕ is strongly event-relativized, then it also holds that (a) $\hat{\gamma}, \tau \not\models \phi$ if $\phi \in \mathit{rel}_\exists$ and (b) $\hat{\gamma}, \tau \models \phi$ if $\phi \in \mathit{rel}_\forall$, for all $\tau \in \mathbb{T} \setminus \{\tau_i \mid i \in \mathbb{N}\}$.*

Observe that the formulas in Example 1 and 2 are not event-relativized. The definition of event-relativized formulas and Theorem 1 straightforwardly extend to richer real-time logics that also contain future operators and are first-order. We point out that most formulas that we encountered when formalizing security policies in such a richer temporal logic are strongly event-relativized [3].

From Theorem 1, it follows that the interval-based semantics can simulate the point-based one by using a fresh event proposition sp with its signal $\gamma_{\mathit{sp}} = \{\tau_i \mid i \in \mathbb{N}\}$, for a time sequence $\bar{\tau}$. We then event-relativize a formula ϕ with the proposition sp, i.e., subformulas of the form $\psi_1 \,\mathsf{S}_I\, \psi_2$ are replaced by $(\mathit{sp} \rightarrow \psi_1) \,\mathsf{S}_I\, (\mathit{sp} \wedge \psi_2)$ and $\psi_1 \,\mathsf{T}_I\, \psi_2$ by $(\mathit{sp} \wedge \psi_1) \,\mathsf{T}_I\, (\mathit{sp} \rightarrow \psi_2)$.

4 Monitoring Algorithms

In this section, we present and analyze our monitoring algorithms for both the point-based and the interval-based setting. Without loss of generality, the algorithms assume that the temporal subformulas of a formula ϕ occur only once in ϕ. Moreover, let P be the set of propositions that occur in ϕ.

4.1 A Point-Based Monitoring Algorithm

Our monitoring algorithm for the point-based semantics iteratively computes the truth values of a formula ϕ at the sample-points $i \in \mathbb{N}$ for a given time

```
step•(ϕ, Γ, τ)
  case ϕ = p
    return p ∈ Γ
  case ϕ = ¬ϕ′
    return not step•(ϕ′, Γ, τ)
  case ϕ = ϕ1 ∧ ϕ2
    return step•(ϕ1, Γ, τ) and step•(ϕ2, Γ, τ)
  case ϕ = ϕ1 S_I ϕ2
    update•(ϕ, Γ, τ)
    if L_ϕ = ⟨⟩ then return false
               else return τ − head(L_ϕ) ∈ I
```

```
init•(ϕ)
  for each ψ ∈ sf(ϕ) with ψ = ψ1 S_I ψ2 do
    L_ψ := ⟨⟩

update•(ϕ, Γ, τ)
  let ϕ1 S_I ϕ2 = ϕ
      b1 = step•(ϕ1, Γ, τ)
      b2 = step•(ϕ2, Γ, τ)
      L = if b1 then drop•(L_ϕ, I, τ) else ⟨⟩
  in if b2 then L_ϕ := L ++ ⟨τ⟩
          else L_ϕ := L
```

Fig. 2. Monitoring in a point-based setting

sequence $\bar{\tau}$ and a family of signals $\hat{\gamma} = (\gamma_p)_{p \in P}$. We point out that $\bar{\tau}$ and $\hat{\gamma}$ are given incrementally, i.e., in the $(i+1)$st iteration, the monitor obtains the time-stamp τ_i and the signals between the previous time-stamp and τ_i. In fact, in the point-based setting, we do not need to consider "chunks" of signals; instead, we can restrict ourselves to the snapshots $\Gamma_i := \{p \in P \mid \tau_i \in \gamma_p\}$, for $i \in \mathbb{N}$, i.e., Γ_i is the set of propositions that hold at time τ_i.

Each iteration of the monitor is performed by executing the procedure $\mathsf{step}^\bullet$. At sample-point $i \in \mathbb{N}$, $\mathsf{step}^\bullet$ takes as arguments the formula ϕ, the snapshot Γ_i, and i's time-stamp τ_i. It computes the truth value of ϕ at i recursively over ϕ's structure. For efficiency, the procedure $\mathsf{step}^\bullet$ maintains for each subformula ψ of the form $\psi_1 \mathsf{S}_I \psi_2$ a sequence L_ψ of time-stamps. These sequences are initialized by the procedure $\mathsf{init}^\bullet$ and updated by the procedure $\mathsf{update}^\bullet$. These three procedures[2] are given in Figure 2 and are described next.

The base case of $\mathsf{step}^\bullet$ where ϕ is a proposition and the cases for the Boolean connectives $\neg$ and $\wedge$ are straightforward. The only involved case is where ϕ is of the form $\phi_1 \mathsf{S}_I \phi_2$. In this case, $\mathsf{step}^\bullet$ first updates the sequence L_ϕ and then computes ϕ's truth value at the sample-point $i \in \mathbb{N}$.

Before we describe how we update the sequence L_ϕ, we describe the elements that are stored in L_ϕ and how we obtain from them ϕ's truth value. After the update of L_ϕ by $\mathsf{update}^\bullet$, the sequence L_ϕ stores the time-stamps τ_j with $\tau_i - \tau_j \in {}^{\leq}I$ (i.e., the time-stamps that satisfy the time constraint now or that might satisfy it in the future) at which ϕ_2 holds and from which ϕ_1 continuously holds up to the current sample-point i (i.e., ϕ_2 holds at $j \leq i$ and ϕ_1 holds at each $k \in \{j+1, \dots, i\}$). Moreover, if there are time-stamps τ_j and $\tau_{j'}$ with $j < j'$ in L_ϕ with $\tau_i - \tau_j \in I$ and $\tau_i - \tau_{j'} \in I$ then we only keep in L_ϕ the time-stamp of the later sample-point, i.e., $\tau_{j'}$. Finally, the time-stamps in L_ϕ are ordered increasingly. Having L_ϕ at hand, it is easy to determine ϕ's truth value. If L_ϕ is the empty sequence then obviously ϕ does not hold at sample-point i. If L_ϕ is non-empty then ϕ holds at i iff the first time-stamp κ in L_ϕ fulfills the timing constraints given by the interval I, i.e., $\tau_i - \kappa \in I$. Recall that ϕ holds at i iff

[2] Our pseudo-code is written in a functional-programming style using pattern matching. $\langle\rangle$ denotes the empty sequence, $+\!\!+$ sequence concatenation, and $x :: L$ the sequence with head x and tail L.

```
drop$^\bullet(L, I, \tau)$
  case $L = \langle\rangle$
    return $\langle\rangle$
  case $L = \kappa :: L'$
    if $\tau - \kappa \notin {}^{\leq}I$ then return drop$^\bullet(L', I, \tau)$
                 else return drop$'^\bullet(\kappa, L', I, \tau)$

drop$'^\bullet(\kappa, L', I, \tau)$
  case $L' = \langle\rangle$
    return $\langle\kappa\rangle$
  case $L' = \kappa' :: L''$
    if $\tau - \kappa' \in I$ then return drop$'^\bullet(\kappa', L'', I, \tau)$
                 else return $\kappa :: L'$
```

Fig. 3. Auxiliary procedures

there is a sample-point $j \leq i$ with $\tau_i - \tau_j \in I$ at which ϕ_2 holds and since then ϕ_1 continuously holds.

Initially, L_ϕ is the empty sequence. If ϕ_2 holds at sample-point i, then update$^\bullet$ adds the time-stamp τ_i to L_ϕ. However, prior to this, it removes the time-stamps of the sample-points from which ϕ_1 does not continuously hold. Clearly, if ϕ_1 does not hold at i then we can empty the sequence L_ϕ. Otherwise, if ϕ_1 holds at i, we first drop the time-stamps for which the distance to the current time-stamp τ_i became too large with respect to the right margin of I. Afterwards, we drop time-stamps until we find the last time-stamp τ_j with $\tau_i - \tau_j \in I$. This is done by the procedures drop$^\bullet$ and drop$'^\bullet$ shown in Figure 3.

Theorem 2. *Let ϕ be a formula, $\hat{\gamma} = (\gamma_p)_{p \in P}$ be a family of signals, $\bar{\tau}$ be a time sequence, and $n > 0$. The procedure* step$^\bullet(\phi, \Gamma_{n-1}, \tau_{n-1})$ *terminates, and returns* **true** *iff $\hat{\gamma}, \bar{\tau}, n-1 \mathrel{\dot{\models}} \phi$, whenever* init$^\bullet(\phi)$, step$^\bullet(\phi, \Gamma_0, \tau_0)$, ..., step$^\bullet(\phi, \Gamma_{n-2}, \tau_{n-2})$ *were called previously in this order, where $\Gamma_i = \{p \in P \mid \tau_i \in \gamma_p\}$, for $i < n$.*

We end this subsection by analyzing the monitor's computational complexity. Observe that we cannot bound the space that is needed to represent the time-stamps in the time sequence $\bar{\tau}$. They become arbitrarily large as time progresses. Moreover, since the time domain is dense, they can be arbitrarily close to each other. As a consequence, operations like subtraction of elements from $\mathbb{T}$ cannot be done in constant time. We return to this point in Section 4.3.

In the following, we assume that each $\tau \in \mathbb{T}$ is represented by two bit strings for the numerator and denominator. The representation of an interval I consists of the representations for $\ell(I)$ and $r(I)$ and whether the left margin and right margin is closed or open. We denote the maximum length of these bit strings by $\|\tau\|$ and $\|I\|$, respectively. The operations on elements in $\mathbb{T}$ that the monitoring algorithm performs are subtractions and membership tests. Subtraction $\tau - \tau'$ can be carried out in time $\mathcal{O}(m^2)$, where $m = \max\{\|\tau\|, \|\tau'\|\}$.[3] A membership test $\tau \in I$ can also be carried out in time $\mathcal{O}(m^2)$, where $m = \max\{\|\tau\|, \|I\|\}$.

The following theorem establishes an upper bound on the time complexity of our monitoring algorithm.

[3] Note that $\frac{p}{q} - \frac{p'}{q'} = \frac{p \cdot q' - p' \cdot q}{q \cdot q'}$ and that $\mathcal{O}(m^2)$ is an upper bound on the multiplication of two m bit integers. There are more sophisticated algorithms for multiplication that run in $\mathcal{O}(m \log m \log\log m)$ time [19] and $\mathcal{O}(m \log m 2^{\log^* m})$ time [8]. For simplicity, we use the quadratic upper bound.

Theorem 3. *Let ϕ, $\hat{\gamma}$, $\bar{\tau}$, n, and $\Gamma_0, \dots, \Gamma_{n-1}$ be as in Theorem 2. Executing the sequence* $\mathsf{init}^\bullet(\phi)$, $\mathsf{step}^\bullet(\phi, \Gamma_0, \tau_0)$, $\dots$, $\mathsf{step}^\bullet(\phi, \Gamma_{n-1}, \tau_{n-1})$ *requires* $\mathcal{O}\big(m^2 \cdot n \cdot |\phi|\big)$ *time, where* $m = \max\big(\{\|I\| \mid \alpha \; \mathsf{S}_I \; \beta \in \mathsf{sf}(\phi)\} \cup \{\|\tau_0\|, \dots, \|\tau_{n-1}\|\}\big)$.

4.2 An Interval-Based Monitoring Algorithm

Our monitoring algorithm for the interval-based semantics determines, for a given family of signals $\hat{\gamma} = (\gamma_p)_{p \in P}$, the truth value of a formula ϕ, for any $\tau \in \mathbb{T}$. In other words, it determines the set $\gamma_{\phi,\hat{\gamma}} := \{\tau \in \mathbb{T} \mid \hat{\gamma}, \tau \models \phi\}$. We simply write γ_ϕ instead of $\gamma_{\phi,\hat{\gamma}}$ when the family of signals $\hat{\gamma}$ is clear from the context. Similar to the point-based setting, the monitor incrementally receives the input $\hat{\gamma}$ and incrementally outputs γ_ϕ, i.e., the input and output signals are split into "chunks" by an infinite interval partition $\bar{J}$. Concretely, the input of the $(i+1)$st iteration consists of the formula ϕ that is monitored, the interval J_i of $\bar{J}$, and the family $\hat{\Delta}_i = (\Delta_{i,p})_{p \in P}$ of sequences of intervals $\Delta_{i,p} = \overline{\mathsf{iip}}^1(\gamma_p \cap J_i)$, for propositions $p \in P$. The output of the $(i+1)$st iteration is the sequence $\overline{\mathsf{iip}}^1(\gamma_\phi \cap J_i)$.

Observe that the sequence $\overline{\mathsf{iip}}^1(\gamma_p \cap J_i)$ only consists of a finite number of intervals since the signal γ_p satisfies the finite-variability condition and J_i is bounded. Moreover, since γ_p is stable on every interval in $\overline{\mathsf{iip}}(\gamma_p)$ and an interval has a finite representation, the sequence $\overline{\mathsf{iip}}^1(\gamma_p \cap J_i)$ finitely represents the signal chunk $\gamma_p \cap J_i$. Similar observations are valid for the signal chunk $\gamma_\phi \cap J_i$.

Each iteration is performed by the procedure step. To handle the since operator efficiently, step maintains for each subformula ψ of the form $\psi_1 \; \mathsf{S}_I \; \psi_2$, a (possibly empty) interval K_ψ and a finite sequence of intervals Δ_ψ. These global variables are initialized by the procedure init and updated by the procedure update. These three procedures are given in Figure 4 and are described next.

The procedure step computes the signal chunk $\gamma_\phi \cap J_i$ recursively over the formula structure. It utilizes the right-hand sides of the following equalities:

$$\gamma_p \cap J_i = \bigcup\nolimits_{K \in \overline{\mathsf{iip}}^1(\gamma_p \cap J_i)} K \tag{1}$$

$$\gamma_{\neg\phi'} \cap J_i = J_i \setminus \Big(\bigcup\nolimits_{K \in \overline{\mathsf{iip}}^1(\gamma_{\phi'} \cap J_i)} K\Big) \tag{2}$$

$$\gamma_{\phi_1 \wedge \phi_2} \cap J_i = \bigcup\nolimits_{\substack{K_1 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_1} \cap J_i) \\ K_2 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap J_i)}} (K_1 \cap K_2) \tag{3}$$

$$\gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i = \bigcup\nolimits_{\substack{K_1 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_1}) \text{ with } K_1 \cap J_i \neq \emptyset \\ K_2 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_2}) \text{ with } (K_2 \oplus I) \cap (J_i^{\geq}) \neq \emptyset}} \Big(\big((K_2 \cap {}^{+}K_1) \oplus I\big) \cap K_1 \cap J_i\Big) \tag{4}$$

where ${}^{+}K := \{\ell(K)\} \cup K$, for $K \in \mathbb{I}$, i.e., making the interval K left-closed.

The equalities (1), (2), and (3) are obvious and their right-hand sides are directly reflected in our pseudo-code. The case where ϕ is a proposition is straightforward. For the case $\phi = \neg\phi'$, we use the procedure invert, shown in Figure 5, to compute $\overline{\mathsf{iip}}^1(\gamma_\phi \cap J_i)$ from $\Delta' = \overline{\mathsf{iip}}^1(\gamma_{\phi'} \cap J_i)$. This is done by "complementing" Δ' with respect to the interval J_i. For instance, the output of $\mathsf{invert}\big(\langle [1,2]\,(3,4)\rangle, [0,10)\big)$ is $\langle [0,1)\,(2,3]\,[4,10)\rangle$. For the case $\phi = \phi_1 \wedge \phi_2$, we use the procedure intersect, also shown in Figure 5, to compute $\overline{\mathsf{iip}}^1(\gamma_\phi \cap J_i)$ from $\Delta_1 = \overline{\mathsf{iip}}^1(\gamma_{\phi_1} \cap J_i)$ and $\Delta_2 = \overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap J_i)$. This procedure returns the

```
step($\phi, \hat{\Delta}, J$)
  case $\phi = p$
    return $\Delta_p$
  case $\phi = \neg\phi'$
    let $\Delta' = \mathsf{step}(\phi', \hat{\Delta}, J)$
    in return $\mathsf{invert}(\Delta', J)$
  case $\phi = \phi_1 \wedge \phi_2$
    let $\Delta_1 = \mathsf{step}(\phi_1, \hat{\Delta}, J)$
        $\Delta_2 = \mathsf{step}(\phi_2, \hat{\Delta}, J)$
    in return $\mathsf{intersect}(\Delta_1, \Delta_2)$
  case $\phi = \phi_1 \mathsf{S}_I \phi_2$
    let $(\Delta'_1, \Delta'_2) = \mathsf{update}(\phi, \hat{\Delta}, J)$
    in return $\mathsf{merge}(\mathsf{combine}(\Delta'_1, \Delta'_2, I, J))$
```

```
init($\phi$)
  for each $\psi \in \mathsf{sf}(\phi)$ with $\psi = \psi_1 \mathsf{S}_I \psi_2$ do
    $K_\psi := \emptyset$
    $\Delta_\psi := \langle\rangle$

update($\phi, \hat{\Delta}, J$)
    let $\phi_1 \mathsf{S}_I \phi_2 = \phi$
        $\Delta_1 = \mathsf{step}(\phi_1, \hat{\Delta}, J)$
        $\Delta_2 = \mathsf{step}(\phi_2, \hat{\Delta}, J)$
        $\Delta'_1 = \mathsf{prepend}(K_\phi, \Delta_1)$
        $\Delta'_2 = \mathsf{concat}(\Delta_\phi, \Delta_2)$
    in $K_\phi :=$ if $\Delta'_1 = \langle\rangle$ then $\emptyset$ else $\mathsf{last}(\Delta'_1)$
       $\Delta_\phi := \mathsf{drop}(\Delta'_2, I, J)$
       return $(\Delta'_1, \Delta'_2)$
```

Fig. 4. Monitoring in an interval-based setting

```
cons($K, \Delta$)
  if $K = \emptyset$ then
    return $\Delta$
  else
    return $K :: \Delta$

invert($\Delta, J$)
  case $\Delta = \langle\rangle$
    return $\langle J \rangle$
  case $\Delta = K :: \Delta'$
    return $\mathsf{cons}(J \cap {}^{<}K, \mathsf{invert}(\Delta', J \cap (K^{>})))$
```

```
intersect($\Delta_1, \Delta_2$)
  if $\Delta_1 = \langle\rangle$ or $\Delta_2 = \langle\rangle$ then
    return $\langle\rangle$
  else
    let $K_1 :: \Delta'_1 = \Delta_1$
        $K_2 :: \Delta'_2 = \Delta_2$
    in if $K_1 \cap (K_2^{\geq}) = \emptyset$ then
         return $\mathsf{cons}(K_1 \cap K_2, \mathsf{intersect}(\Delta'_1, \Delta_2))$
       else
         return $\mathsf{cons}(K_1 \cap K_2, \mathsf{intersect}(\Delta_1, \Delta'_2))$
```

Fig. 5. The auxiliary procedures for the Boolean connectives

sequence of intervals that have a non-empty intersection of two intervals in the input sequences. The elements in the returned sequence are ordered increasingly.

The equality (4) for $\phi = \phi_1 \mathsf{S}_I \phi_2$ is less obvious and using its right-hand side for an implementation is also less straightforward since the intervals K_1 and K_2 are not restricted to occur in the current chunk J_i. Instead, they are intervals in $\overline{\mathsf{iip}}^1(\gamma_{\phi_1})$ and $\overline{\mathsf{iip}}^1(\gamma_{\phi_2})$, respectively, with certain constraints.

Before giving further implementation details, we first show why equality (4) holds. To prove the inclusion $\subseteq$, assume $\tau \in \gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i$. By the semantics of the since operator, there is a $\tau_2 \in \gamma_{\phi_2}$ with $\tau - \tau_2 \in I$ and $\tau_1 \in \gamma_{\phi_1}$, for all $\tau_1 \in (\tau_2, \tau]$.

- Obviously, $\tau_2 \in K_2$, for some $K_2 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_2})$. By taking the time constraint I into account, K_2 satisfies the constraint $(K_2 \oplus I) \cap (J_i^{\geq}) \neq \emptyset$. Note that even the more restrictive constraint $(K_2 \oplus I) \cap J_i \neq \emptyset$ holds. However, we employ the weaker constraint in our implementation as it is useful for later iterations.
- Since $\overline{\mathsf{iip}}(\gamma_{\phi_1})$ is the coarsest interval partition of γ_{ϕ_1}, there is an interval $K_1 \in \overline{\mathsf{iip}}^1(\gamma_{\phi_1})$ with $(\tau_2, \tau] \subseteq K_1$. As $\tau \in J_i$, the constraint $K_1 \cap J_i \neq \emptyset$ holds.

It follows that $\tau \in K_1$ and $\tau_2 \in {}^{+}K_1$, and thus $\tau_2 \in K_2 \cap {}^{+}K_1$. From $\tau - \tau_2 \in I$, we obtain that $\tau \in (K_2 \cap {}^{+}K_1) \oplus I$. Finally, since $\tau \in K_1 \cap J_i$, we have that $\tau \in ((K_2 \cap {}^{+}K_1) \oplus I) \cap K_1 \cap J_i$. The other inclusion $\supseteq$ can be shown similarly.

```
prepend($K, \Delta$)
  if $K = \emptyset$ then
    return $\Delta$
  else
    case $\Delta = \langle\rangle$
      return $\langle K \rangle$
    case $\Delta = K' :: \Delta'$
      if adjacent($K, K'$) or $K \cap K' \neq \emptyset$ then
        return $K \cup K' :: \Delta'$
      else
        return $K :: \Delta$
```

```
combine($\Delta_1', \Delta_2', I, J$)
  if $\Delta_1' = \langle\rangle$ or $\Delta_2' = \langle\rangle$ then return $\langle\rangle$
  else
    let $K_2 :: \Delta_2'' = \Delta_2'$
    in if $(K_2 \oplus I) \cap J = \emptyset$ then return $\langle\rangle$
       else
         let $K_1 :: \Delta_1'' = \Delta_1'$
               $\Delta =$ if $K_2^{>} \cap {}^{+}K_1 = \emptyset$ then
                          combine($\Delta_1'', \Delta_2', I, J$)
                        else
                          combine($\Delta_1', \Delta_2'', I, J$)
         in return $(K_2 \cap {}^{+}K_1) \oplus I) \cap K_1 \cap J :: \Delta$
```

```
concat($\Delta_1, \Delta_2$)
  case $\Delta_1 = \langle\rangle$
    return $\Delta_2$
  case $\Delta_1 = \Delta_1' +\!\!+ \langle K_1 \rangle$
    return $\Delta_1' +\!\!+$ prepend($K_1, \Delta_2$)
```

```
merge($\Delta$)
  case $\Delta = \langle\rangle$
    return $\Delta$
  case $\Delta = K :: \Delta'$
    return prepend($K$, merge($\Delta'$))
```

```
drop($\Delta_2', I, J$)
  case $\Delta_2' = \langle\rangle$
    return $\langle\rangle$
  case $\Delta_2' = K_2 :: \Delta_2''$
    let $K = (K_2 \oplus I) \cap (J^{>})$
    in if $K = \emptyset$ then return drop($\Delta_2'', I, J$)
                        else return drop'($K, \Delta_2', I, J$)
```

```
drop'($K, \Delta_2', I, J$)
  case $\Delta_2' = \langle\rangle$
    return $\langle K \rangle$
  case $\Delta_2' = K_2 :: \Delta_2''$
    let $K' = (K_2 \oplus I) \cap (J^{>})$
    in if $K \subseteq K'$ then return drop'($K', \Delta_2'', I, J$)
                         else return $\Delta_2'$
```

Fig. 6. The auxiliary procedures for the since operator

For computing the signal chunk $\gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i$, the procedure step first determines the subsequences Δ_1' and Δ_2' of $\overline{\text{ıɪp}}^1(\gamma_{\phi_1})$ and $\overline{\text{ıɪp}}^1(\gamma_{\phi_2})$ consisting of those intervals K_1 and K_2 appearing in the equality (4), respectively. This is done by the procedure update. Afterwards, step computes the sequence $\overline{\text{ıɪp}}^1(\gamma_\phi \cap J_i)$ from Δ_1' and Δ_2' by using the procedures combine and merge, given in Figure 6. We now explain how merge(combine($\Delta_1', \Delta_2', I, J$)) returns the sequence $\overline{\text{ıɪp}}^1(\gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i)$. First, combine($\Delta_1', \Delta_2', I, J$) computes a sequence of intervals whose union is $\gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i$. It traverses the ordered sequences Δ_1' and Δ_2' and adds the interval $((K_2 \cap {}^{+}K_1) \oplus I) \cap K_1 \cap J_i$ to the resulting ordered sequence, for K_1 in Δ_1' and K_2 in Δ_2'. The test $K_2^{>} \cap {}^{+}K_1 = \emptyset$ determines in which sequence (Δ_1' or Δ_2') we advance next: if the test succeeds then $K_2' \cap {}^{+}K_1 = \emptyset$ where K_2' is the successor of K_2 in Δ_2', and hence we advance in Δ_1'. The sequence Δ_2' is not necessarily entirely traversed: when $(K_2 \oplus I) \cap J_i = \emptyset$, one need not inspect other elements K_2' of the sequence Δ_2', as then $((K_2' \cap {}^{+}K_1) \oplus I) \cap K_1 \cap J_i = \emptyset$. The elements in the sequence returned by the combine procedure might be empty, adjacent, or overlapping. The merge procedure removes empty elements and merges adjacent or overlapping intervals, i.e., it returns the sequence $\overline{\text{ıɪp}}^1(\gamma_{\phi_1 \mathsf{S}_I \phi_2} \cap J_i)$.

Finally, we explain the contents of the variables K_ϕ and Δ_ϕ and how they are updated. We start with K_ϕ. At the $(i+1)$st iteration, for some $i \geq 0$, the following invariant is satisfied by K_ϕ: before the update, the interval K_ϕ is the last interval of $\overline{\text{ıɪp}}^1(\gamma_{\phi_1} \cap {}^{\leq}J_{i-1})$ if $i > 0$ and this sequence is not empty, and K_ϕ is the empty set otherwise. The interval K_ϕ is prepended to the sequence $\overline{\text{ıɪp}}^1(\gamma_{\phi_1} \cap J_i)$ using the prepend procedure from Figure 6, which merges K_ϕ with the first interval of $\Delta_1 = \overline{\mathsf{iip}}^1(\gamma_{\phi_1} \cap J_i)$ if these two intervals are adjacent. The obtained sequence Δ_1' is the maximal subsequence of $\overline{\mathsf{iip}}^1(\gamma_{\phi_1} \cap {}^{\leq}J_i)$ such that $K_1 \cap J_i \neq \emptyset$, for each interval K_1 in Δ_1'. Thus, after the update, K_ϕ is the last interval of $\overline{\mathsf{iip}}^1(\gamma_{\phi_1} \cap {}^{\leq}J_i)$ if this sequence is not empty, and K_ϕ is the empty set otherwise. Hence the invariant on K_ϕ is preserved at the next iteration.

The following invariant is satisfied by Δ_ϕ at the $(i+1)$st iteration: before the update, the sequence Δ_ϕ is empty if $i = 0$, and otherwise, if $i > 0$, it stores the intervals K_2 in $\overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap {}^{\leq}J_{i-1})$ with $(K_2 \oplus I) \cap (J_{i-1}^{>}) \neq \emptyset$ and $(K_2 \oplus I) \cap (J_{i-1}^{>}) \not\subseteq (K_2' \oplus I) \cap (J_{i-1}^{>})$, where K_2' is the successor of K_2 in $\overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap {}^{\leq}J_{i-1})$. The procedure concat concatenates the sequence Δ_ϕ with the sequence $\Delta_2 = \overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap J_i)$. Since the last interval of Δ_ϕ and the first interval of Δ_2 can be adjacent, concat might need to merge them. Thus, the obtained sequence Δ_2' is a subsequence of $\overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap {}^{\leq}J_i)$ such that $(K_2 \oplus I) \cap (J_i^{\geq}) \neq \emptyset$, for each element K_2. Note that $J_{i-1}^{>} = J_i^{\geq}$. The updated sequence Δ_ϕ is obtained from Δ_2' by removing the intervals K_2 with $(K_2 \oplus I) \cap (J_i^{>}) = \emptyset$, i.e., the intervals that are irrelevant for later iterations. The procedure drop from Figure 6 removes these intervals. Moreover, if there are intervals K_2 and K_2' in Δ_ϕ with $(K_2 \oplus I) \cap (J_i^{>}) \subseteq (K_2' \oplus I) \cap (J_i^{>})$ then only the interval that occurs later is kept in Δ_ϕ. This is done by the procedure drop'. Thus, after the update, the sequence Δ_ϕ stores the intervals K_2 in $\overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap {}^{\leq}J_i)$ with $(K_2 \oplus I) \cap (J_i^{>}) \neq \emptyset$ and $(K_2 \oplus I) \cap (J_i^{>}) \not\subseteq (K_2' \oplus I) \cap (J_i^{>})$, where K_2' is the successor of K_2 in $\overline{\mathsf{iip}}^1(\gamma_{\phi_2} \cap {}^{\leq}J_i)$. Hence the invariant on Δ_ϕ is preserved at the next iteration.

Theorem 4. *Let ϕ be a formula, $\hat{\gamma} = (\gamma_p)_{p \in P}$ a family of signals, $\bar{J}$ an infinite interval partition, and $n > 0$. The procedure $\mathsf{step}(\phi, \hat{\Delta}_{n-1}, J_{n-1})$ terminates and returns the sequence $\overline{\mathsf{iip}}^1(\gamma_\phi \cap J_{n-1})$, whenever $\mathsf{init}(\phi)$, $\mathsf{step}(\phi, \hat{\Delta}_0, J_0)$, ..., $\mathsf{step}(\phi, \hat{\Delta}_{n-2}, J_{n-2})$ were called previously in this order, where $\hat{\Delta}_i = (\Delta_{i,p})_{p \in P}$ with $\Delta_{i,p} = \overline{\mathsf{iip}}^1(\gamma_p \cap J_i)$, for $i < n$.*

Finally, we analyze the monitor's computational complexity. As in the point-based setting, we take the representation size of elements of the time domain $\mathbb{T}$ into account. The basic operations here in which elements of $\mathbb{T}$ are involved are operations on intervals like checking emptiness (i.e. $I = \emptyset$), "extension" (e.g. $I^{>}$), and "shifting" (i.e. $I \oplus J$). The representation size of the interval $I \oplus J$ is in $\mathcal{O}(\|I\| + \|J\|)$. The time to carry out the shift operation is in $\mathcal{O}(\max\{\|I\|, \|J\|\}^2)$. All the other basic operations that return an interval do not increase the representation size of the resulting interval with respect to the given intervals. However, the time complexity is quadratic in the representation size of the given intervals whenever the operation needs to compare interval margins.

The following theorem establishes an upper bound on the time complexity of our monitoring algorithm.

Theorem 5. *Let ϕ, $\hat{\gamma}$, $\bar{J}$, n, and $\hat{\Delta}_i$ be given as in Theorem 4. Executing the sequence $\mathsf{init}(\phi)$, $\mathsf{step}(\phi, \hat{\Delta}_0, J_0)$, ..., $\mathsf{step}(\phi, \hat{\Delta}_{n-1}, J_{n-1})$ requires $\mathcal{O}\big(m^2 \cdot (n + \delta \cdot |\phi|) \cdot |\phi|^3\big)$ time, where $m = \max\big(\{\|I\| \mid \alpha \,\mathsf{S}_I\, \beta \in \mathsf{sf}(\phi)\} \cup \{\|J_0\|, \ldots, \|J_{n-1}\|\} \cup \bigcup_{p \in P}\{\|K\| \mid K \in \overline{\mathsf{iip}}^1(\gamma_p \cap ({}^{<}J_n))\}\big)$ and $\delta = \sum_{p \in P} \|\gamma_p \cap ({}^{<}J_n)\|$.*

We remark that the factor $m^2 \cdot |\phi|^2$ is due to the operations on the margins of intervals. With the assumption that the representation of elements of the time domain is constant, we obtain the upper bound $\mathcal{O}\big((n + \delta \cdot |\phi|) \cdot |\phi|\big)$.

4.3 Time Domains

The stated worst-case complexities of both monitoring algorithms take the representation size of the elements in the time domain into account. In practice, it is often reasonable to assume that these elements have a bounded representation, since arbitrarily precise clocks do not exist. For example, for many applications it suffices to represent time-stamps as Unix time, i.e., 32 or 64 bit signed integers. The operations performed by our monitoring algorithms on the time domain elements would then be carried out in constant time. However, a consequence of this practically motivated assumption is that the time domain is discrete and bounded rather than dense and unbounded.

For a discrete time domain, we must slightly modify the interval-based monitoring algorithm, namely, the operator ${}^{+}K$ used in the equality (4) must be redefined. In a discrete time domain, we extend K by one point in time to the left if it exists, i.e., ${}^{+}K := K \cup \{k - 1 \mid k \in K \text{ and } k > 0\}$. No modifications are needed for the point-based algorithm. If we assume a discrete and unbounded time domain, we still cannot assume that the operations on elements from the time domain can be carried out in constant time. But multiplication is no longer needed to compare elements in the time domain and thus the operations can be carried in time linear in the representation size. The worst-case complexity of both algorithms improves accordingly.

When assuming limited-precision clocks, which results in a discrete time domain, a so-called fictitious-clock semantics [2, 18] is often used. This semantics formalizes, for example, that if the system event e happens strictly before the event e' but both events fall between two clock ticks, then we can distinguish them by temporal ordering, not by time. In a fictitious-clock semantics, we time-stamp e and e' with the same clock value and in a trace e appears strictly before e'. For ordering e and e' in a trace, signals must be synchronized. Our point-based monitoring algorithm can directly be used for a fictitious-clock semantics. It iteratively processes a sequence of snapshots $\langle \Gamma_0, \Gamma_1, \dots \rangle$ together with a sequence of time-stamps $\langle \tau_0, \tau_1, \dots \rangle$, which is increasing but not necessarily strictly increasing anymore. In contrast, our interval-based monitoring algorithm does not directly carry over to a fictitious-clock semantics.

4.4 Comparison of the Monitoring Algorithms

In the following, we compare our two algorithms when monitoring a strongly event-relativized formula ϕ. By Theorem 1, the point-based setting and the interval-based setting coincide on this formula class.

First note that the input for the $(i+1)$th iteration of the point-based monitoring algorithm can be easily obtained online from the given signals $\hat{\gamma} = (\gamma)_{p \in S \cup E}$. Whenever an event occurs, we record the time $\tau_i \in \mathbb{T}$, determine the current truth values of the propositions, i.e., $\Gamma_i = \{p \in P \mid \tau_i \in \gamma_p\}$, and invoke the monitor by executing $\mathsf{step}^\bullet(\phi, \Gamma_i, \tau_i)$. The worst-case complexity of the point-based monitoring algorithm of the first n iterations is $\mathcal{O}\big(m^2 \cdot n \cdot |\phi|\big)$, where m is according to Theorem 3.

When using the interval-based monitoring algorithm, we are more flexible in that we need not invoke the monitoring algorithm whenever an event occurs. Instead, we can freely split the signals into chunks. Let $\bar{J}$ be a splitting in which the n'th interval $J_{n'-1}$ is right-closed and $r(J_{n'-1}) = \tau_{n-1}$. We have the worst-case complexity of $\mathcal{O}\big(m'^2 \cdot (n' + \delta \cdot |\phi|) \cdot |\phi|^3\big)$, where m' and δ are according to Theorem 5. We can lower this upper bound, since the formula ϕ is strongly event-relativized. Instead of the factor $m'^2 \cdot |\phi|^2$ for processing the interval margins in the n' iterations, we only have the factor m'^2. The reason is that the margins of the intervals in the signal chunks of subformulas of the form $\psi_1 \mathsf{S}_I \psi_2$ already appear as interval margins in the input.

Note that $m' \geq m$ and that δ is independent of n'. Under the assumption that $m' = m$, the upper bounds on the running times for different splittings only differ by n', i.e., how often we invoke the procedure step. The case where $n' = 1$ corresponds to the scenario where we use the monitoring algorithm offline (up to time τ_{n-1}). The case where $n' = n$ corresponds to the case where we invoke the monitor whenever an event occurs. Even when using the interval-based monitoring algorithm offline and assuming constant representation of the elements in $\mathbb{T}$, the upper bounds differ by the factors n and $\delta \cdot |\phi|$. Since $\delta \geq n$, the upper bound of the point-based monitoring algorithm is lower. In fact, there are examples showing that the gap between the running times matches our upper bounds and that $\delta \cdot |\phi|$ can be significantly larger than n.

5 Related Work

We only discuss the monitoring algorithms most closely related to ours, namely, those of Basin et al. [4], Thati and Roşu [20], and Nickovic and Maler [14, 15].

The point-based monitoring algorithms here simplify and optimize the monitoring algorithm of Basin et al. [4] given for the future-bounded fragment of metric first-order temporal logic. We restricted ourselves here to the propositional setting and to the past-only fragment of metric temporal logic to compare the effect of different time models on monitoring.

Thati and Roşu [20] provide a monitoring algorithm for metric temporal logic with a point-based semantics, which uses formula rewriting. Their algorithm is more general than ours for the point-based setting since it handles past and future operators. Their complexity analysis is based on the assumption that operations involving elements from the time domain can be carried out in constant time. The worst-case complexity of their algorithm on the past-only fragment is worse than ours, since rewriting a formula can generate additional formulas. In particular, their algorithm is not linear in the number of subformulas.

Nickovic and Maler's [14, 15] monitoring algorithms are for the interval-based setting and have ingredients similar to our algorithm for this setting. These ingredients were first presented by Nickovic and Maler for an offline version of their monitoring algorithms [13] for the fragment of interval metric temporal logic with bounded future operators. Their setting is more general in that their signals are continuous functions and not Boolean values for each point in time. Moreover, their algorithms also handle bounded [15] and unbounded [14] future operators by delaying the evaluation of subformulas. The algorithm in [14] slightly differs from the one in [15]: [14] also handles past operators and before starting monitoring, it rewrites the given formula to eliminate the temporal operators until and since with timing constraints. The main difference to our algorithm is that Maler and Nickovic do not provide algorithmic details for handling the Boolean connectives and the temporal operators. In fact, the worst-case complexity, which is only stated for their offline algorithm [13], seems to be too low even when ignoring representation and complexity issues for elements of the time domain.

We are not aware of any work that compares different time models for runtime verification. The surveys [2, 6, 16] on real-time logics focus on expressiveness, satisfiability, and automatic verification of real-time systems. A comparison of a point-based and interval-based time model for temporal databases with a discrete time domain is given by Toman [21]. The work by Furia and Rossi [9] on sampling and the work on digitization [11] by Henzinger et al. are orthogonal to our comparison. These relate fragments of metric interval temporal logic with respect to a discrete and a dense time domain.

6 Conclusions

We have presented, analyzed, and compared monitoring algorithms for real-time logics with point-based and interval-based semantics. Our comparison provides a detailed explanation of trade-offs between the different time models with respect to monitoring. Moreover, we have presented a practically relevant fragment for the interval-based setting by distinguishing between state variables and system events, which can be more efficiently monitored in the point-based setting.

As future work, we plan to extend the monitoring algorithms to handle bounded future operators. This includes analyzing their computational complexities and comparing them experimentally. Another line of research is to establish lower bounds for monitoring real-time logics. Thati and Roşu [20] give lower bounds for future fragments of metric temporal logic including the next operator. However, we are not aware of any lower bounds for the past-only fragment.

References

1. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)
2. Alur, R., Henzinger, T.: Logics and Models of Real Time: A Survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
3. Basin, D., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: SACMAT 2010, pp. 23–33 (2010)
4. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: FSTTCS 2008, pp. 49–60 (2008)

5. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of Real-Time Properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006)
6. Bouyer, P.: Model-checking times temporal logics. In: 5th Workshop on Methods for Modalities. ENTCS, vol. 231, pp. 323–341 (2009)
7. Drusinsky, D.: On-line monitoring of metric temporal logic with time-series constraints using alternating finite automata. J. UCS 12(5), 482–498 (2006)
8. Fürer, M.: Faster integer multiplication. In: STOC 2007, pp. 55–67 (2007)
9. Furia, C., Rossi, M.: A theory of sampling for continuous-time metric temporal logic. ACM Trans. Comput. Log. 12(1) (2010)
10. Goodloe, A., Pike, L.: Monitoring distributed real-time systems: A survey and future directions. Tech. rep. CR-2010-216724, NASA Langley Research Center (2010)
11. Henzinger, T., Manna, Z., Pnueli, A.: What Good are Digital Clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 545–558. Springer, Heidelberg (1992)
12. Kristoffersen, K., Pedersen, C., Andersen, H.: Runtime verification of timed LTL using disjunctive normalized equation systems. In: RV 2003. ENTCS, vol. 89, pp. 210–225 (2003)
13. Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
14. Ničković, D.: Checking Timed and Hybrid Properties: Theory and Applications. PhD thesis, Université Joseph Fourier, Grenoble, France (2008)
15. Nickovic, D., Maler, O.: AMT: A Property-Based Monitoring Tool for Analog Systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007)
16. Ouaknine, J., Worrell, J.: Some Recent Results in Metric Temporal Logic. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 1–13. Springer, Heidelberg (2008)
17. Pike, L., Goodloe, A., Morisset, R., Niller, S.: Copilot: A Hard Real-Time Runtime Monitor. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 345–359. Springer, Heidelberg (2010)
18. Raskin, J.-F., Schobbens, P.-Y.: Real-time Logics: Fictitious Clock as an Abstraction of Dense Time. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 165–182. Springer, Heidelberg (1997)
19. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing 7(3-4), 281–292 (1971)
20. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. In: RV 2004. ENTCS, vol. 113, pp. 145–162 (2005)
21. Toman, D.: Point vs. interval-based query languages for temporal databases. In: PODS 1996, pp. 58–67 (1996)