Title

Robust Online Monitoring of Signal Temporal Logic

Permalink

https://escholarship.org/uc/item/7dz7r2rm

Authors

Deshmukh, Jyotirmoy V
Donzé, Alexandre
Ghosh, Shromona
et al.

Publication Date

2015

DOI

10.1007/978-3-319-23820-3_4

Robust Online Monitoring of Signal Temporal Logic

Jyotirmoy V. Deshmukh[1], Alexandre Donzé[2], Shromona Ghosh[2]
Xiaoqing Jin[1], Garvit Juniwal[2], and Sanjit A. Seshia[2]

[1] Toyota Technical Center, firstname.lastname@tema.toyota.com
[2] University of California Berkeley,
{donze, shromona.ghosh, garvitjuniwal, sseshia}@eecs.berkeley.edu

Abstract. Signal Temporal Logic (STL) is a formalism used to rigorously specify requirements of cyberphysical systems (CPS), i.e., systems mixing digital or discrete components in interaction with a continuous environment or analog components. STL is naturally equipped with a quantitative semantics which can be used for various purposes: from assessing the robustness of a specification to guiding searches over the input and parameter space with the goal of falsifying the given property over system behaviors. Algorithms have been proposed and implemented for *offline* computation of such quantitative semantics, but only few methods exist for an *online* setting, where one would want to monitor the satisfaction of a formula during simulation. In this paper, we formalize a semantics for robust online monitoring of *partial* traces, i.e., traces for which there might not be enough data to decide the Boolean satisfaction (and to compute its quantitative counterpart). We propose an efficient algorithm to compute it and demonstrate its usage on two large scale real-world case studies coming from the automotive domain and from CPS education in a Massively Open Online Course (MOOC) setting. We show that savings in computationally expensive simulations far outweigh any overheads incurred by an online approach.

1 Introduction

Design engineers for embedded control software typically validate their designs by inspecting concrete observations of system behavior. For instance, in the model-based development (MBD) paradigm, designers have access to numerical simulation tools to obtain traces from models of systems. An important problem is then to be able to efficiently test whether some logical property φ holds for a given simulation trace. It is increasingly common [13, 9, 12, 2, 15] to specify such properties using a real-time temporal logic such as Signal Temporal Logic (STL) [7] or Metric Temporal Logic (MTL) [10]. An *offline monitoring* approach involves performing an *a posteriori* analysis on *complete* simulation traces (i.e., traces starting at time 0, and lasting till a user-specified time horizon). Theoretical and practical results for offline monitoring [10, 5, 7, 17] focus on the efficiency of monitoring as a function of the length of the trace, and the size of the formula representing the property φ.

There are a number of situations where offline monitoring is unsuitable. Consider the case where the monitor is to be deployed in an actual system to detect erroneous behavior. As embedded software is typically resource constrained, offline monitoring – which requires storing the entire observed trace – is impractical. Also, when a monitor is used in a simulation-based validation tool, a single simulation may run for several minutes or even hours. If we wish to monitor a safety property over the simulation, a better use of resources is to abort the simulation whenever a violation is detected. Such situations demand an *online monitoring algorithm*, which has markedly different requirements. In particular, a good online monitoring algorithm must: (1) be able to generate intermediate estimates of property satisfaction based on *partial signals*, (2) use minimal amount of data storage, and (3) be able to run fast enough in a real-time setting.

Most works on online monitoring algorithms for logics such as Linear Temporal Logic (LTL) or Metric Temporal Logic (MTL) have focussed on the Boolean satisfaction of properties by partial signals [11, 8, 18]. However, recent work has shown that by assigning quantitative semantics to real-time logics such as MTL and STL, problems such as bug-finding, parameter synthesis, and robustness analysis can be solved using powerful off-the-shelf optimization tools [1, 4]. A robust satisfaction value is a function mapping a property φ and a trace $\mathbf{x}(t)$ to a real number. A large positive value suggests that $\mathbf{x}(t)$ easily satisfies φ, a positive value close to zero suggests that $\mathbf{x}(t)$ is close to violating φ, and a negative value indicates a violation of φ. While the recursive definitions of quantitative semantics naturally define offline monitoring algorithms to compute robust satisfaction values [10, 7, 5], there is limited work on an online monitoring algorithm to do the same [3].

The main technical and theoretical challenge of online monitoring lies in the definition of a practical semantics for a temporal logic formula over a partial signal, i.e., a signal trace with incomplete data which cannot yet validate or invalidate φ. Past work [8] has identified three views for the satisfaction of a LTL property φ over a partial trace τ: (1) a *weak view* where the truth value of φ over τ is assigned to *true* if there is some suffix of τ that satisfies φ, (2) a *strong view* when it is defined to be *false* when some suffix of τ does not satisfy φ and (3) a *neutral view* when the truth value is defined using a truncated semantics of LTL restricted to *finite* paths. In [11], the authors extend the truncated semantics to MTL, and in [3], the authors introduce the notion of a *predictor*, which works as an oracle to complete the partial trace and provide an estimated satisfaction value. However, such a value cannot be formally trusted in general as long as the data is incomplete.

We now outline our major contributions in this paper. In Section 3, we present *robust interval* semantics for an STL property φ on a partial trace τ that unifies the different semantic views of real-time logics on truncated paths. Informally, the robust interval semantics map a trace $\mathbf{x}(t)$ and an STL property φ to an interval (ℓ, υ), with the interpretation that for any suffix $u(t)$, ℓ is the greatest lower bound on the quantitative semantics of the trace $\mathbf{x}(t)$, and υ is the corresponding lowest upper bound. There is a natural correspondence between the interval semantics and three-valued semantics: (1) the truth value of φ is false according to the weak view iff υ is negative, and true otherwise; (2) the truth value is true according to the strong view iff ℓ is positive, and false otherwise; and (3) a neutral semantics, e.g., based on some predictor, can be defined when $\ell < 0 < \upsilon$, i.e., when there exist both suffixes that can violate or satisfy φ.

In Section 4, we present an efficient online algorithm to compute the robust interval semantics for bounded horizon formulas. Our approach is based on the offline algorithm of [5] extended to work in a fashion similar to the incremental Boolean monitoring of STL implemented in the tool AMT [18]. A key feature of our algorithm is that it imposes minimal runtime overhead with respect to the offline algorithm, while being able to compute robust satisfaction intervals on partial traces. In Section 5, we present specialized algorithms to deal with commonly-used unbounded horizon formulas using only a bounded amount of memory.

Finally, we present an implementation and experimental results on two large-scale case studies: (i) industrial-scale Simulink models from the automotive domain in Section 6, and (ii) an automatic grading system used in a massive online education initiative on CPS [14]. Since the online algorithm can abort simulation as soon as the truth value of the property is determined, we see a consistent 10%-20% savings in simulation time (which is typically several hours) in a majority of experiments, with negligible overhead ($< 1\%$). In general, our results indicate that the benefits of our online monitoring algorithm over the offline approach far outweigh any overheads.

2 Background

Interval Arithmetic. We now review interval arithmetic. An interval I is a convex subset of $\mathbb{R}$. A singular interval $[a, a]$ contains exactly one point. Intervals (a, a), $[a, a)$, $(a, a]$, and $\emptyset$ denote empty intervals. We enumerate interval operations below assuming open intervals. Similar operations can be defined for closed, open-closed, and closed-open intervals.

$$\begin{array}{llll}
1.\ -I_1 & = (-b_1, -a_1) & 3.\ I_1 \oplus I_2 & = (a_1 + a_2, b_1 + b_2) \\
2.\ c + I_1 & = (c + a_1, c + b_1) & 4.\ \min(I_1, I_2) & = (\min(a_1, a_2), \min(b_1, b_2))
\end{array} \tag{2.1}$$

$$5.\ I_1 \cap I_2 = \begin{cases} \emptyset & \text{if } \min(b_1, b_2) < \max(a_1, a_2) \\ (\max(a_1, a_2), \min(b_1, b_2)) & \text{otherwise.} \end{cases}$$

Definition 1 (Signal). *A* time domain $\mathcal{T}$ *is a finite or infinite set of time instants such that* $\mathcal{T} \subseteq \mathbb{R}^{\geq 0}$ *with* $0 \in \mathcal{T}$. *A* signal $\mathbf{x}$ *is a function from* $\mathcal{T}$ *to* $\mathcal{X}$. *Given a time domain* $\mathcal{T}$, *a* partial signal *is any signal defined on a time domain* $\mathcal{T}' \subseteq \mathcal{T}$.

Simulation frameworks typically provide signal values at discrete time instants, usually this is a by-product of using a numerical technique to solve the differential equations in the underlying system. These discrete-time solutions are assumed to be sampled versions of the actual signal, which can be reconstructed using some form of interpolation. In this paper, we assume constant interpolation to reconstruct the signal $\mathbf{x}(t)$, i.e., given a sequence of time-value pairs $(t_0, \mathbf{x}_0), \ldots, (t_n, \mathbf{x}_n)$, for all $t \in [t_0, t_n)$, we define $\mathbf{x}(t) = \mathbf{x}_i$ if $t \in [t_i, t_{i+1})$, and $\mathbf{x}(t_n) = \mathbf{x}_n$. Further, let $\mathcal{T}_n \subseteq \mathcal{T}$ represent the finite subset of time instants at which the signal values are given.

Signal Temporal Logic. We use Signal Temporal Logic (STL) [7] to analyze time-varying behaviors of signals. We now present its syntax and semantics. A *signal predicate* μ is a formula of the form $f(\mathbf{x}) > 0$, where $\mathbf{x}$ is a variable that takes values from $\mathcal{X}$, and f is a function from $\mathcal{X}$ to $\mathbb{R}$. For a given f, let f_{inf} denote $\inf_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$, i.e., the *greatest lower bound* of f over $\mathcal{X}$. Similarly, let $f_{\mathsf{sup}} = \sup_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$. The syntax of an STL formula φ is defined in Eq. (2.2). Note that $\Box$ and $\Diamond$ can be defined in terms of the $\mathbf{U}$ operator, but we include them for convenience.

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_{(u,v)}\varphi \mid \Diamond_{(u,v)}\varphi \mid \varphi \mathbf{U}_{(u,v)}\varphi \tag{2.2}$$

Quantitative semantics for timed-temporal logics have been proposed for STL in [7]; we include the definition below.

Definition 2 (Robust Satisfaction Value). *The* robust satisfaction value *is a function ρ mapping φ, the signal $\mathbf{x}$, and a time $\tau \in \mathcal{T}$ as follows:*

$$\begin{array}{lcl}
\rho\left(f(\mathbf{x}) > 0, \mathbf{x}, \tau\right) & = & f(\mathbf{x}(\tau)) \\
\rho\left(\neg\varphi, \mathbf{x}, \tau\right) & = & -\rho(\varphi, \mathbf{x}, \tau) \\
\rho\left(\varphi_1 \wedge \varphi_2, \mathbf{x}, \tau\right) & = & \min\left(\rho(\varphi_1, \mathbf{x}, \tau), \rho(\varphi_2, \mathbf{x}, \tau)\right) \\
\rho\left(\Box_I \varphi, \mathbf{x}, \tau\right) & = & \inf\limits_{\tau' \in \tau + I} \rho(\varphi, \mathbf{x}, \tau') \\
\rho\left(\Diamond_I \varphi, \mathbf{x}, \tau\right) & = & \sup\limits_{\tau' \in \tau + I} \rho(\varphi, \mathbf{x}, \tau') \\
\rho\left(\varphi \mathbf{U}_I \psi, \mathbf{x}, \tau\right) & = & \sup\limits_{\tau_1 \in \tau + I} \min\left(\rho(\psi, \mathbf{x}, \tau_1), \inf\limits_{\tau_2 \in (\tau, \tau_1)} \rho(\varphi, \mathbf{x}, \tau_2)\right)
\end{array} \tag{2.3}$$

Here, the translation from quantitative semantics to the usual Boolean satisfaction semantics is that a signal $\mathbf{x}$ satisfies an STL formula φ at a time τ iff the robust satisfaction value $\rho(\varphi, \mathbf{x}, \tau) \geq 0$.

3 Robust Interval Semantics

In what follows, we assume that we wish to monitor the robust satisfaction value of a signal over a finite time-horizon T_H. We assume that the signal is obtained by applying piecewise constant interpolation to a sampled signal defined over time-instants $\{t_0, t_1, \ldots, t_N\}$, such that $t_N = T_H$. In an online monitoring context, at any time t_i, only the partial signal over time instants $\{t_0, \ldots, t_i\}$ is available, and the rest of the signal becomes available in discrete time increments. We define robust satisfaction semantics of STL formulas over such partial signals using an interval-based semantics. Such a *robust satisfaction interval* (`RoSI`) includes all possible robust satisfaction values corresponding to the suffixes of the partial signal. In this section, we formalize the recursive definitions for the robust satisfaction interval of an STL formula with respect to a partial signal, and in the next section we will discuss an efficient algorithm to compute and maintain these intervals.

Definition 3 (Prefix, Completions). *Let $\{t_0, \ldots, t_i\}$ be a finite set of time instants such that $t_i \leq T_H$, and let $\mathbf{x}_{[0,i]}$ be a partial signal over the time domain $[t_0, t_i]$. We say that $\mathbf{x}_{[0,i]}$ is a prefix of a signal $\mathbf{x}$ if for all $t \leq t_i$, $\mathbf{x}(t) = \mathbf{x}_{[0,i]}(t)$. The* set of completions *of a partial signal $\mathbf{x}_{[0,i]}$ (denoted by $\mathcal{C}(\mathbf{x}_{[0,i]})$) is defined as the set $\{\mathbf{x} \mid \mathbf{x}_{[0,i]}$ is a prefix of $\mathbf{x}\}$.*

Definition 4 (Robust Satisfaction Interval (`RoSI`)). *The robust satisfaction interval of an STL formula φ on a partial signal $\mathbf{x}_{[0,i]}$ at a time $\tau \in [t_0, t_N]$ is an interval I such that:*

$$\inf(I) = \inf_{\mathbf{x} \in \mathcal{C}(\mathbf{x}_{[0,i]})} \rho(\varphi, \mathbf{x}, \tau) \qquad \text{and} \qquad \sup(I) = \sup_{\mathbf{x} \in \mathcal{C}(\mathbf{x}_{[0,i]})} \rho(\varphi, \mathbf{x}, \tau)$$

Definition 5. *We now define a recursive function $[\rho]$ that maps a given formula φ, a partial signal $\mathbf{x}_{[0,i]}$ and a time $\tau \in \mathcal{T}$ to an interval $[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)$.*

$$
\begin{array}{lcl}
[\rho]\left(f(\mathbf{x}_{[0,i]}) > 0, \mathbf{x}_{[0,i]}, \tau\right) & = & \begin{cases} [f(\mathbf{x}_{[0,i]}(\tau)), f(\mathbf{x}_{[0,i]}(\tau))] & \tau \in [t_0, t_i] \\ [f_{\mathsf{inf}}, f_{\mathsf{sup}}] & \textit{otherwise.} \end{cases} \\
[\rho]\left(\neg\varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & -[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau) \\
[\rho]\left(\varphi_1 \wedge \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) & = & \min([\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau), [\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau)) \\
[\rho]\left(\Box_I \varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & \inf\limits_{t \in \tau + I} \left([\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)\right) \\
[\rho]\left(\Diamond_I \varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & \sup\limits_{t \in \tau + I} \left([\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)\right) \\
[\rho]\left(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) & = & \sup\limits_{\tau_2 \in \tau + I} \min \left(\begin{array}{l} [\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau_2), \\ \inf\limits_{\tau_1 \in (\tau, \tau_2)} [\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau_1)) \end{array} \right)
\end{array} \tag{3.1}
$$

The following lemma that can be proved by induction over the structure of STL formulas shows that the interval obtained by applying the recursive definition for $[\rho]$ is indeed the robust satisfaction interval as defined in Def. 4.

Lemma 1. *For any STL formula φ, the function $[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)$ defines the robust satisfaction interval for the formula φ over the signal $\mathbf{x}_{[0,i]}$ at time τ.*

4 Online Algorithm

Donzé et al. [5] present an offline algorithm for monitoring STL formulas over (piecewise) linearly interpolated signals. A naïve implementation of an online algorithm is as follows: at time t_i, use a modification of the offline monitoring algorithm to recursively compute the robust satisfaction intervals as defined by Def. 5 to the signal $\mathbf{x}_{[0,i]}$. We observe that such a procedure does many repeated computations that can be avoided by maintaining the results of intermediate computations. Furthermore, the naïve procedure requires storing the signal values over the entire time horizon, which makes it memory-intensive. In this section, we present the main technical contribution of this paper: *an online algorithm that is memory-efficient and avoids repeated computations*.

As in the offline monitoring algorithm in [5], an essential ingredient of the online algorithm is Lemire's running maximum filter algorithm [16]. The problem this algorithm addresses is the following: given a sequence of values $a_1, \ldots, a_n$, find the maximum (resp. minimum) over all windows of size w, i.e., for all j, $\max_{i \in [j, j+w)} a_i$ (resp. $\min_{i \in [j, j+w)} a_i$). We briefly review an extension of Lemire's algorithm over piecewise-constant signals with variable time steps, given as Algorithm 1. The main observation in Lemire's algorithm is that it is sufficient to maintain a descending (resp. ascending) monotonic edge (noted `F` in Algorithm 1) to compute the sliding maxima (resp. minima), in order to achieve an optimal procedure (measured in terms of the number of comparisons between elements).

We first focus on the fragment of STL where each temporal operator is bounded by a time-interval I such that $\sup(I)$ is finite. The procedure for online monitoring is an algorithm that maintains in memory the syntax tree of the formula φ to be monitored, augmented with some book-keeping information. First, we formalize some notation. For a given formula φ, let $\mathcal{T}_\varphi$ represent the syntax tree of φ, and let $\mathsf{root}(\mathcal{T}_\varphi)$ denote the root of the tree. Each node in the syntax tree (other than a leaf node) corresponds to

```
Algorithm 1: SlidingMax($(t_0, \mathbf{x}_0), \ldots, (t_N, \mathbf{x}_N)$)
Input: Window: $[a, b]$
Output: Sliding maximum $\mathbf{y}(t)$ over times in $[t_0, t_N]$
F := $\{0\}$ // F is the set of times representing the monotonic edge
$i := 0$ ; $s, t := t_0 - b$
while $t + a < t_N$ do
    if F $\neq \emptyset$ then $t := \min(t_{\min(\mathtt{F})} - a, t_{i+1} - b)$ else $t := t_{i+1} - b$ if $t = t_{i+1} - b$ then
        while $\mathbf{x}_{i+1} \geq \mathbf{x}_{\max(\mathtt{F})} \wedge$ F $\neq \emptyset$ do
            F:= F $- \max$(F)
        F:= F $\cup \{i+1\}$, $i := i + 1$
    else// Slide window to the right
        if $s > t_0$ then $\mathbf{y}(s) := \mathbf{x}_{\min(\mathtt{F})}$ else $\mathbf{y}(t_0) := \mathbf{x}_{\min(\mathtt{F})}$ F:= F $- \min$(F), $s := t$
```

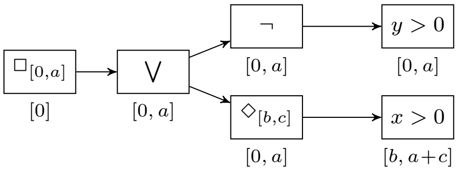


Fig. 1. Syntax tree $\mathcal{T}_\varphi$ for φ (given in (4.2)) with each node v annotated with $\mathsf{hor}(v)$.

an STL operator $\neg, \vee, \wedge, \Box_I$ or $\Diamond_I$.[3] We will use $\mathbf{H}_I$ to denote any temporal operator bounded by interval I. For a given node v, let $\mathsf{op}(v)$ denote the operator for that node. For any node v in $\mathcal{T}_\varphi$ (except the root node), let $\mathsf{parent}(v)$ denote the unique parent of v.

Algorithm 2 does the online RoSI computation. Like the offline algorithm, it is a dynamic programming algorithm operating on the syntax tree of the given STL formula, i.e., computation of the RoSI of a formula combines the RoSIs for its constituent subformulas in a bottom-up fashion. As computing the RoSI at a node v requires the RoSIs at the child-nodes, this computation has to be delayed till the RoSIs at the children of v in a certain time-interval are available. We call this time-interval the *time horizon* of v (denoted $\mathsf{hor}(v)$), and define it recursively in Eq. (4.1).

$$\mathsf{hor}(v) = \begin{cases} [0] & \text{if } v = \mathsf{root}(\mathcal{T}_\varphi) \\ I \oplus \mathsf{hor}(\mathsf{parent}(v)) & \text{if } v \neq \mathsf{root}(\mathcal{T}_\varphi) \text{ and } \mathsf{op}(\mathsf{parent}(v)) = \mathbf{H}_I \\ \mathsf{hor}(\mathsf{parent}(v)) & \text{otherwise.} \end{cases} \quad (4.1)$$

We illustrate the working of the algorithm using a small example then give a brief sketch of the various steps in the algorithm.

Example 1. Consider formula (4.2). We show $\mathcal{T}_\varphi$ and $\mathsf{hor}(v)$ for each node v in $\mathcal{T}_\varphi$ in Fig. 1. In rest of the paper, we use φ as a running example[4].

$$\varphi \triangleq \Box_{[0,a]} \big(\neg(y > 0) \vee \Diamond_{[b,c]}(x > 0)\big) \quad (4.2)$$

[3] We omit the case of $\mathbf{U}_I$ here for lack of space, although the rewriting approach of [5] can also be adapted and was implemented in our tool.

[4] We remark that φ is equivalent to $\Box_{[0,a]} \big((y > 0) \implies \Diamond_{[b,c]}(x > 0)\big)$, which is a common formula used to express a timed causal relation between two signals.

The algorithm augments each node v of $\mathcal{T}_\varphi$ with a double-ended queue, that we denote $\mathsf{worklist}[v]$. Let ψ be the subformula denoted by the tree rooted at v. For the partial signal $\mathbf{x}_{[0,i]}$, the algorithm maintains in $\mathsf{worklist}[v]$, the $\texttt{RoSI}$ $[\rho](\psi, \mathbf{x}_{[0,i]}, t)$ for each $t \in \mathsf{hor}(v) \cap [t_0, t_i]$. We denote by $\mathsf{worklist}[v](t)$ the entry corresponding to time t in $\mathsf{worklist}[v]$. When a new data-point $\mathbf{x}_{i+1}$ corresponding to the time t_{i+1} is available, the monitoring procedure updates each $[\rho](\psi, \mathbf{x}_{[0,i]}, t)$ in $\mathsf{worklist}[v]$ to $[\rho](\psi, \mathbf{x}_{[0,i+1]}, t)$.

In Fig. 3, we give an example of a run of the algorithm. We assume that the algorithm starts in a state where it has processed the partial signal $\mathbf{x}_{[0,2]}$, and show the effect of receiving data at time-points t_3, t_4 and t_5. The figure shows the states of the worklists at each node of $\mathcal{T}_\varphi$ at these times when monitoring the STL formula φ presented in Eq. (4.2). Each row in the table adjacent to a node shows the state of the worklist after the algorithm processes the value at the time indicated in the first column.

The first row of the table shows the snapshot of the worklists at time t_2. Observe that in the worklists for the subformula $y > 0$, $\neg y > 0$, because $a < b$, the data required to compute the $\texttt{RoSI}$ at t_0, t_1 and the time a, is available, and hence each of the $\texttt{RoSIs}$ is singular. On the other hand, for the subformula $x > 0$, the time horizon is $[b, a + c]$, and no signal value is available at any time in this interval. Thus, at time t_2, all elements of $\mathsf{worklist}[v_{x>0}]$ are $(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$ corresponding to the greatest lower bound and lowest upper bound on x.

To compute the values of $\Diamond_{[b,c]}(x > 0)$ at any time t, we take the supremum over values from times $t + b$ to $t + c$. As the time horizon for the node corresponding to $\Diamond_{[b,c]}(x > 0)$ is $[0, a]$, t ranges over $[0, a]$. In other words, we wish to perform the sliding maximum over the interval $[0 + b, a + c]$, with a window of length $c - b$. We can use the algorithm for computing the sliding window maximum as discussed earlier in this section. One caveat is that we need to store separate monotonic edges for the upper and lower bounds of the $\texttt{RoSIs}$. The algorithm then proceeds upward on the syntax tree, only updating the worklist of a node only when there is an update to the worklists of its children.

The second row in each table is the effect of obtaining a new time point (at time t_3) for both signals. Note that this does not affect $\mathsf{worklist}[v_{y>0}]$ or $\mathsf{worklist}[v_{\neg y>0}]$, as all $\texttt{RoSIs}$ are already singular, but does update the $\texttt{RoSI}$ values for the node $v_{x>0}$. The algorithm then invokes Alg. 1 on $\mathsf{worklist}[v_{x>0}]$ to update $\mathsf{worklist}[v_{\Diamond_{[b,c]}(x>0)}]$. Note that in the invocation on the second row (corresponding to time t_3), there is an additional value in the worklist, at time t_3. This leads Alg. 1 to produce a new value of $\mathsf{SlidingMax}\left(\mathsf{worklist}[v_{\mathbf{x}>0}], [b, c]\right)(t_3 - b)$, which is then inserted in $\mathsf{worklist}[v_{\Diamond_{[b,c]} x>0}]$. This leads to additional points appearing in worklists at the ancestors of this node.

Finally, we remark that the run of this algorithm shows that at time t_4, the $\texttt{RoSI}$ for the formula φ is $[-2, -2]$, which yields a negative upper bound, showing that the formula is not satisfied irrespective of the suffixes of x and y. In other words, the satisfaction of φ is known before we have all the data required by $\mathsf{hor}(\varphi)$.

Alg. 2 is essentially a procedure that recursively visits each node in the syntax tree $\mathcal{T}_\varphi$ of the STL formula φ that we wish to monitor. Line 4 corresponds to the base case of the recursion, i.e. when the algorithm visits a leaf of $\mathcal{T}_\varphi$ or an atomic predicates of the form $f(\mathbf{x}) > 0$. Here, the algorithm inserts the pair $(t_{i+1}, \mathbf{x}_{i+1})$ in $\mathsf{worklist}[v_{f(\mathbf{x})>0}]$ if t_{i+1} lies inside $\mathsf{hor}(v_{f(\mathbf{x})>0})$. In other words, it only tracks a value if it is useful for the computing the robust satisfaction interval of some ancestor node.

For a node corresponding to a Boolean operation, the algorithm first updates the worklists at the children, and then uses them to update the worklist at the node. If

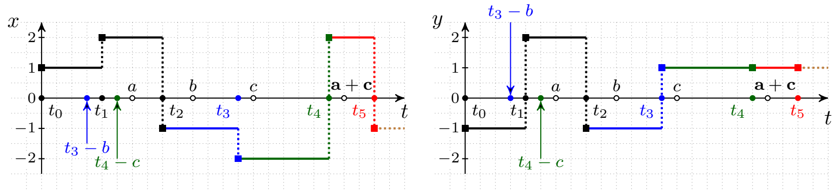
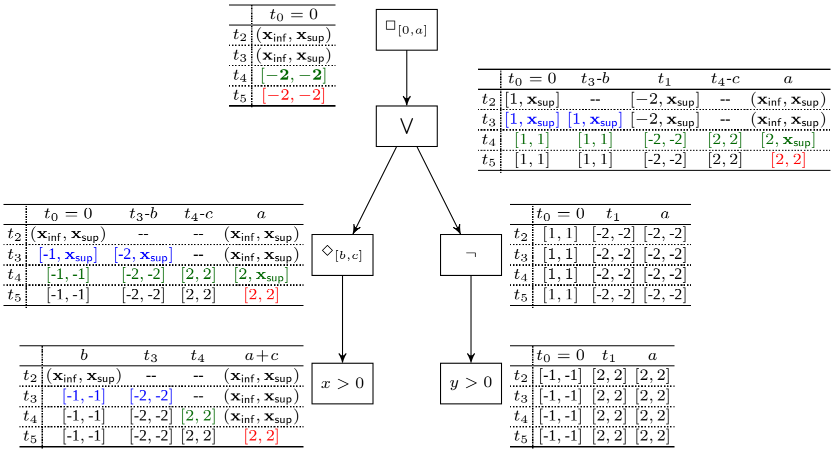


Fig. 2. These plots show the signals $x(t)$ and $y(t)$. Each signal begins at time $t_0 = 0$, and we consider three partial signals: $\mathbf{x}_{[0,3]}$ (black + blue), and $\mathbf{x}_{[0,4]}$ ($\mathbf{x}_{[0,3]}$ + green), and $\mathbf{x}_{[0,5]}$ ($\mathbf{x}_{[0,4]}$ + red).



	$t_0 = 0$
t_2	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_3	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_4	$[\mathbf{-2}, \mathbf{-2}]$
t_5	$[-2, -2]$

	$t_0 = 0$	t_3-b	t_1	t_4-c	a
t_2	$[1, \mathbf{x}_{\mathsf{sup}}]$	--	$[-2, \mathbf{x}_{\mathsf{sup}}]$	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_3	$[1, \mathbf{x}_{\mathsf{sup}}]$	$[1, \mathbf{x}_{\mathsf{sup}}]$	$[-2, \mathbf{x}_{\mathsf{sup}}]$	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_4	[1, 1]	[1, 1]	[-2, -2]	[2, 2]	$[2, \mathbf{x}_{\mathsf{sup}}]$
t_5	[1, 1]	[1, 1]	[-2, -2]	[2, 2]	[2, 2]

	$t_0 = 0$	t_3-b	t_4-c	a
t_2	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$	--	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_3	$[-1, \mathbf{x}_{\mathsf{sup}}]$	$[-2, \mathbf{x}_{\mathsf{sup}}]$	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_4	[-1, -1]	[-2, -2]	[2, 2]	$[2, \mathbf{x}_{\mathsf{sup}}]$
t_5	[-1, -1]	[-2, -2]	[2, 2]	[2, 2]

	$t_0 = 0$	t_1	a
t_2	[1, 1]	[-2, -2]	[-2, -2]
t_3	[1, 1]	[-2, -2]	[-2, -2]
t_4	[1, 1]	[-2, -2]	[-2, -2]
t_5	[1, 1]	[-2, -2]	[-2, -2]

	b	t_3	t_4	$a+c$
t_2	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$	--	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_3	[-1, -1]	[-2, -2]	--	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_4	[-1, -1]	[-2, -2]	[2, 2]	$(\mathbf{x}_{\mathsf{inf}}, \mathbf{x}_{\mathsf{sup}})$
t_5	[-1, -1]	[-2, -2]	[2, 2]	[2, 2]

	$t_0 = 0$	t_1	a
t_2	[-1, -1]	[2, 2]	[2, 2]
t_3	[-1, -1]	[2, 2]	[2, 2]
t_4	[-1, -1]	[2, 2]	[2, 2]
t_5	[-1, -1]	[2, 2]	[2, 2]

Fig. 3. We show a snapshot of the $\mathsf{worklist}[v]$ maintained by the algorithm for four different (incremental) partial traces of the signals $x(t)$ and $y(t)$. Each row indicates the state of $\mathsf{worklist}[v]$ at the time indicated in the first column. An entry marked -- indicates that the corresponding element did not exist in $\mathsf{worklist}[v]$ at that time. Each colored entry indicates that the entry was affected by availability of a signal fragment of the corresponding color.

the current node represents $\neg\varphi$ (Line 5), the algorithm flips the sign of each entry in $\mathsf{worklist}[v_\varphi]$; this operation is denoted as $-\mathsf{worklist}[v_\varphi]$. Consider the case where the current node v_ψ is a conjunction $\varphi_1 \wedge \varphi_2$. The sequence of upper bounds and the sequence of lower bounds of the entries in $\mathsf{worklist}[v_{\varphi_1}]$ and $\mathsf{worklist}[v_{\varphi_1}]$ can be each thought of as a piecewise-constant signal (likewise for $\mathsf{worklist}[v_{\varphi_2}]$). In Line 11, the algorithm computes a pointwise-minimum over piecewise-constant signals representing the upper and lower bounds of the `RoSI`s of its arguments. Note that if for $i = 1, 2$, if $\mathsf{worklist}[v_{\varphi_i}]$ has N_i entries, then the pointwise-$\min$ would have to be performed at most $N_1 + N_2$ distinct time-points. Thus, $\mathsf{worklist}[v_{\varphi_1 \wedge \varphi_2}]$ has at most $N_1 + N_2$ entries. A similar phenomenon can be seen in Fig. 3, where computing a $\max$ over the worklists of $v_{\Diamond_{[b,c]}(x>0)}$ and $v_{\neg(y>0)}$ leads to an increase in the number of entries in the worklist of the disjunction.

For nodes corresponding to temporal operators, e.g., $\Diamond_I \varphi$, the algorithm first updates $\mathsf{worklist}[v_\varphi]$. It then applies Alg. 1 to compute the sliding maximum over $\mathsf{worklist}[v_\varphi]$. Note that if $\mathsf{worklist}[v_\varphi]$ contains N entries, so does $\mathsf{worklist}[v_{\Diamond_I \varphi}]$.

Algorithm 2: updateWorkList(v_ψ, t_{i+1}, $\mathbf{x}_{i+1}$)

```
// $v_\psi$ is a node in the syntax tree, $(t_{i+1}, \mathbf{x}_{i+1})$ is a new
   signal time-point
switch $\psi$ do
    case $f(\mathbf{x}) > 0$
        if $t_{i+1} \in \mathsf{hor}(v_\psi)$ then
            worklist$[v_\psi](t_{i+1}) := [f(\mathbf{x}_{i+1}), f(\mathbf{x}_{i+1})]$
    case $\neg\varphi$
        updateWorkList($v_\varphi$, $t_{i+1}$ ,$\mathbf{x}_{i+1}$)
        worklist$[v_\psi] := -$worklist$[v_\varphi]$
    case $\varphi_1 \wedge \varphi_2$
        updateWorkList($v_{\varphi_1}$, $t_{i+1}$, $\mathbf{x}_{i+1}$)
        updateWorkList($v_{\varphi_2}$, $t_{i+1}$, $\mathbf{x}_{i+1}$)
        worklist$[v_\psi] := \min($worklist$[v_{\varphi_1}]$, worklist$[v_{\varphi_2}])$
    case $\Box_I \varphi$
        updateWorkList($v_\varphi$, $t_{i+1}$ , $\mathbf{x}_{i+1}$)
        worklist$[v_\psi] :=$ SlidingMax(worklist$[v_\varphi]$, $I$)
```

A further optimization can be implemented on top of this basic scheme. For a node v corresponding to the subformula $\mathbf{H}_I\varphi$, the first few entries of worklist$[v]$ (say up to time u) could become singular intervals once the required RoSIs for worklist$[v_\varphi]$ are available. The optimization is to only compute SlidingMax over worklist$[v_\varphi]$ starting from $u + \inf(I)$. We omit the pseudo-code for brevity.

5 Monitoring untimed formulas

If the STL formula being monitored has untimed (i.e. infinite-horizon) temporal operators, a direct application of Alg. 2 requires every node in the sub-tree rooted at the untimed operator to have an unbounded time horizon. In other words, for all such nodes, the algorithm would have to keep track of every value over arbitrarily long intervals. For certain untimed operators and the combinations thereof, we show that we can monitor the formulas using only a bounded amount of information.

First, we introduce some equivalences over intervals a, b, c that we use in the theorem and the proof to follow:

$$\min(\max(a,b), \max(a,c)) = \max(a, \min(b,c)) \tag{5.1}$$

$$\min(a, \max(b,c)) = \max(\min(a,b), \min(a,c)) \tag{5.2}$$

$$\max(\max(a,b), c) = \max(a,b,c) \tag{5.3}$$

$$\min(\max(a,b), a) = a \tag{5.4}$$

Theorem 1. *For each of the following formulae, where φ and ψ are atomic predicates of the form $f(\mathbf{x}) > 0$, we can monitor interval robustness in an online fashion using constant memory: (1) $\Box\varphi$, $\Diamond\varphi$, (2) $\varphi\mathbf{U}\psi$, (3) $\Box(\varphi \vee \Diamond\psi)$, $\Diamond(\varphi \wedge \Box\psi)$, (4) $\Box\Diamond\varphi$, $\Diamond\Box\varphi$, and (5) $\Diamond(\varphi \wedge \Diamond\psi)$, $\Box(\varphi \vee \Box\psi)$.*

Proof. In what follows, we use the following short-hand notation:

$$p_i \equiv [\rho](f(\mathbf{x})>0, \mathbf{x}_{[0,n+1]}, t_i) \qquad q_i \equiv [\rho](g(\mathbf{x})>0, \mathbf{x}_{[0,n+1]}, t_i) \tag{5.5}$$

Note that if $i \in [0,n]$, then p_i is the same over the partial signal $\mathbf{x}_{[0,n]}$, i.e., $p_i = [\rho](f(\mathbf{x})>0, \mathbf{x}_{[0,n]}, t_i)$ (and respectively for q_i). We will use this equivalence in several of the steps in what follows.

(1) $\Box\varphi$, where $\varphi \equiv f(\mathbf{x}) > 0$. Observe the following:

$$[\rho](\varphi, \mathbf{x}_{[0,n+1]}, 0) = \min_{i\in[0,n+1]} p_i = \min\left(\min_{i\in[0,n]} p_i,\ p_{n+1}\right) \tag{5.6}$$

In the final expression above, observe that the first entry does not contain any p_{n+1} terms, i.e., it can be computed using the data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the partial signal $\mathbf{x}_{[0,n]}$ itself. Thus, for all n, if we maintain the one interval representing the $\min$ of the first n values of $f(\mathbf{x})$ as a *summary*, then we can compute the interval robustness of $\Box(f(\mathbf{x})>0)$ over $\mathbf{x}_{[0,n+1]}$ with the additional data $\mathbf{x}_{n+1}$ available at t_{n+1}. Note for the dual formula $\Diamond(f(\mathbf{x})>0)$, a similar result holds with $\min$ substituted by $\max$.

(2) $\varphi\mathbf{U}\psi$, where $\varphi \equiv f(\mathbf{x})>0$, and $\psi \equiv g(\mathbf{x})>0$. Observe the following:

$$[\rho](\varphi\mathbf{U}\psi, \mathbf{x}_{[0,n+1]}, 0) = \max_{i\in[0,n+1]} \min(q_i,\ \min_{j\in[0,i]} p_j) \tag{5.7}$$

We can rewrite the RHS of Eq. (5.7) to get:

$$\max\left(\underline{\max_{i\in[0,n]} \min\left(q_i,\ \min_{j\in[0,i]} p_j\right)},\quad \min\left(\underline{\min_{j\in[0,n]}} p_j,\ p_{n+1}, q_{n+1}\right)\right) \tag{5.8}$$

Let U_n and M_n respectively denote the first and second underlined terms in the above expression. Note that for any n, U_n and M_n can be computed only using data $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Consider the recurrences $M_{n+1} = \min(M_n, p_{n+1}, q_{n+1})$ and $U_{n+1} = \max(U_n, M_{n+1})$; we can observe that to compute M_{n+1} and U_{n+1}, we only need M_n, U_n, and $\mathbf{x}_{n+1}$. Furthermore, U_{n+1} is the desired interval robustness value over the partial signal $\mathbf{x}_{[0,n+1]}$. Thus storing and iteratively updating the two interval-values U_n and M_n is enough to monitor the given formula.

(3) $\Box(\varphi \vee \Diamond\psi)$, where $\varphi \equiv f(\mathbf{x})>0$, and $\psi \equiv g(\mathbf{x})>0$. Observe the following:

$$\begin{aligned}
[\rho](\Box(\varphi \vee \Diamond\psi), \mathbf{x}_{[0,n+1]}, 0) &= \min_{i\in[0,n+1]} \max\left(p_i,\ \max_{j\in[i,n+1]} q_j\right) \\
&= \min_{i\in[0,n+1]} \max\left(p_i,\ \max_{j\in[i,n]} q_j, q_{n+1}\right)
\end{aligned} \tag{5.9}$$

Repeatedely applying the equivalence (5.1) to the outer $\min$ in (5.9) we get:

$$\max\left(q_{n+1},\ \min_{i\in[0,n+1]} \max\left(p_i,\ \max_{j\in[i,n]} q_j\right)\right) \tag{5.10}$$

The inner $\min$ simplifies to:

$$\max\left(q_{n+1}, \min\left(p_{n+1},\ \underline{\min_{i\in[0,n]}\left(\max\left(p_i,\ \max_{j\in[i,n]} q_j\right)\right)}\right)\right) \tag{5.11}$$

Let T_n denote the underlined term; note that we do not require any data at time t_{n+1} to compute it. Using the recurrence $T_{n+1} = \max\left(q_{n+1}, \min\left(p_{n+1}, T_n\right)\right)$, we can obtain the desired interval robustness value. The memory required is that for storing the one interval value T_n. A similar result can be established for the dual formula $\Diamond(f(\mathbf{x}) > 0 \wedge \Box(g(\mathbf{x})>0))$.

(4) $\Box\Diamond(\varphi)$, where $\varphi \equiv f(\mathbf{x})>0$. Observe the following:

$$[\rho](\Box\Diamond(\varphi, \mathbf{x}_{[0,n+1]}, 0) = \min_{i\in[0,n+1]} \max_{j\in[i,n+1]} p_j \tag{5.12}$$

Rewriting the outer $\min$ operator and the inner $\max$ more explicitly, we get:

$$\min\left(\underline{\min_{i\in[0,n]}\max\left(\max_{j\in[i,n]} p_j, p_{n+1}\right),}\quad p_{n+1}\right) \tag{5.13}$$

Repeatedly using (5.1) to simplify the above underlined term we get:

$$\min\left(\max\left(p_{n+1}, \min_{i\in[0,n]}\max_{j\in[i,n]} p_j\right), p_{n+1}\right) = p_{n+1}. \tag{5.14}$$

The simplification to p_{n+1}, follows from (5.4). Thus, to monitor $\Box\Diamond(f(\mathbf{x})>0)$, we do not need to store any information, as the interval robustness simply evaluates to that of the predicate $f(\mathbf{x}) > 0$ at time t_{n+1}. A similar result can be obtained for the dual formula $\Diamond\Box(f(\mathbf{x}) >> 0)$.

(5) $\Diamond(\varphi \wedge \Diamond(\psi))$, where $\varphi \equiv f(\mathbf{x})>0$ $\psi \equiv \Diamond(g(\mathbf{x})>0))$. Observe the following:

$$[\rho](\Diamond(\varphi\wedge\Diamond(\psi)), \mathbf{x}_{[0,n+1]}, 0) = \max_{i\in[0,n+1]}\left(\min\left(p_i, \max_{j\in[i,n+1]} q_j\right)\right) \tag{5.15}$$

We can rewrite the RHS of Eq. (5.15) as the first expression below. Applying the equivalence in (5.2) and (5.3) to the expression on the left, we get the expression on the right.

$$\max\left(\begin{array}{l}\min\left(p_0, \max\left(q_0,\ldots,q_{n+1}\right)\right)\\ \ldots\\ \min\left(p_n, \max\left(q_n, q_{n+1}\right)\right)\\ \min\left(p_{n+1}, q_{n+1}\right)\end{array}\right) = \max\left(\begin{array}{l}\min(p_0,q_0),\ldots,\min(p_0,q_{n+1}),\\ \ldots\\ \min(p_n,q_n),\min(p_n,q_{n+1}),\\ \min(p_{n+1},q_{n+1})\end{array}\right) \tag{5.16}$$

Grouping terms containing q_{n+1} together and applying the equivalence in (5.2) we get:

$$\max\left(\begin{array}{l}\max\left(\begin{array}{l}\min(p_0,q_0),\min(p_0,q_1),\ldots,\min(p_0,q_n),\\ \min(p_1,q_1),\ldots,\min(p_1,q_n),\\ \ldots\\ \min(p_n,q_n)\end{array}\right),\\ \min(q_{n+1},\underline{\max(p_0,p_1,\ldots,p_n)}),\\ \min(p_{n+1},q_{n+1})\end{array}\right) \tag{5.17}$$

Observe that the first argument to the outermost $\max$ can be computed using only $\mathbf{x}_1,\ldots,\mathbf{x}_n$. Suppose we denote this term T_n. Also note that in the second argument, the inner $\max$ (underlined) can be computed using only $\mathbf{x}_1,\ldots,\mathbf{x}_n$. Let us denote this term by M_n. We now have a recurrence relations:

$$M_{n+1} = \max(M_n, p_{n+1}), \tag{5.18}$$

$$T_{n+1} = \max(T_n, \min(q_{n+1}, M_n), \min(q_{n+1}, p_{n+1})), \tag{5.19}$$

where T_0 = $\min(p_0, q_0)$ and M_0 = p_0. Thus, the desired interval robustness can be computed using only two values stored in T_n and M_n. The dual result holds for the formula $\Box(\varphi \vee \Box(\psi))$.

Remarks on extending above result: The result in Theorem 1 can be generalized to allow φ and ψ that are not atomic predicates, under following two conditions:

1. Bounded horizon subformulae condition: For each formula, the subformulae φ and ψ have a bounded time-horizon, i.e., $\mathsf{hor}(\varphi)$ and $\mathsf{hor}(\psi)$ are closed intervals.

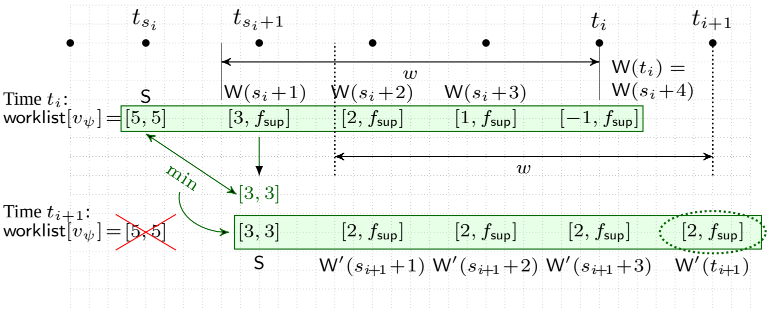


Fig. 4. A depiction of the action of the procedure to update the summary while computing $[\rho](\Box\psi, \mathbf{x}_{[0,i]}, t_0)$. Here, $\mathsf{W}(j)$ is shorthand for $[\rho](\psi, \mathbf{x}_{[0,i]}, t_j)$ and $\mathsf{W}'(j)$ is shorthand for $[\rho](\psi, \mathbf{x}_{[0,i+1]}, t_j)$.

2. Smallest step-size condition: Consecutive time-points in the signal are at least $\mathit{\Delta}$ seconds apart, for some finite $\mathit{\Delta}$, which is known *a priori*.

5.1 Generalizing Theorem 1

Let $\mathsf{sub}(\varphi)$ denote the set of all subformulas of φ except φ itself. Let $\mathsf{last}(\varphi)$ be defined as follows:

$$\mathsf{last}(\varphi) \triangleq \max_{\psi \in \mathsf{sub}(\varphi)} \sup(\mathsf{hor}(\psi)) \tag{5.20}$$

The meaning of $\mathsf{last}(\varphi)$ is as follows: the last time at which a data value of $\mathbf{x}$ is required to compute $\rho(\varphi, \mathbf{x}, t)$, is $t + \mathsf{last}(\varphi)$. For the formula φ defined in Eq. (4.2), $\mathsf{last}(\varphi) = a + c$. For the formula $\psi \equiv \Box(x > 0)$, $\mathsf{last}(\psi) = \infty$. In general, for any untimed formula φ, $\mathsf{last}(\varphi)$ is equal to ∞. In Theorem 1, we show that certain classes of untimed formulas can be monitored in an online fashion with bounded amount of memory. We first define the following quantities:

$$\mathit{\Delta} \triangleq \min_{i \geq 0}(t_{i+1} - t_i) \qquad w_\varphi \triangleq \max_{\psi \in \mathsf{sub}(\varphi)} \mathsf{last}(\psi) \qquad k_\varphi \triangleq \left\lceil \frac{w_\varphi}{\mathit{\Delta}} \right\rceil. \tag{5.21}$$

Here, $\mathit{\Delta}$ represents the smallest time-step in the monitored signal, w_φ is the largest time horizon of all subformulas of φ, and k_φ is the largest number of discrete time-points for the trace in any w_φ interval.

Theorem 2. *If w_φ is finite, then for each φ listed below, we can monitor* `RoSI` *of φ in an online fashion using $O(k_\varphi)$ memory.*

1. $\Box\psi$ *(dually $\Diamond\psi$)*
2. $\varphi \mathbf{U} \psi$
3. $\Box\Diamond\psi$ *(dually $\Diamond\Box\psi$)*
4. $\Box(\varphi \vee \Diamond\psi)$*(dually $\Diamond(\varphi \wedge \Box\psi)$),*
5. $\Diamond(\varphi \wedge \Diamond\psi)$ *(dually $\Box(\varphi \vee \Diamond\psi)$*

(5.22)

Proof. We provide proof sketches. The main argument in each of the proofs is as follows: For any partial signal $\mathbf{x}_{[0,i]}$, there are two cases: The first case is when $t_0 \geq t_i - w_\varphi$. By assumption, there are at most k_φ time-points in the interval $[t_0, t_i]$. Thus, in this case,

Algorithm 3: Computing `RoSI` for untimed Until

```
$v_1 := \mathsf{S}_\varphi, v_2 := \mathsf{S}_\psi$
foreach $j \in [s_i + 1, i]$ do
    $v_1 := \min\big(v_1, [\rho](\varphi, \mathbf{x}_{[0,i]}, t_j)\big)$
    $v_2 := \sup\big(v_2, \min\big(v_1, [\rho](\psi, \mathbf{x}_{[0,i]}, t_j)\big)\big)$
$[\rho](\varphi \mathbf{U} \psi, \mathbf{x}_{[0,i]}, t_0) := v_2$
```

the worklists at each of the nodes v_ψ corresponding to $\psi \in \mathsf{sub}(\varphi)$ have to track at most k_φ `RoSI` values in order to compute $[\rho](\varphi, \mathbf{x}, t_0)$.

The second case is when $t_0 < t_i - w_\varphi$; this implies that there is a largest time t_{s_i} in $[t_0, t_1, \ldots, t_i]$ such that $t_{s_i} < t_i - w_\varphi$. For the partial signal $\mathbf{x}_{[0,i]}$, at each time $t \leq t_{s_i}$, there is enough information to compute the exact robustness value of each of the subformulas of φ. The central step is that for each of the formulas mentioned above, the robustness values in the interval $[t_0, t_{s_i}]$ can be *summarized* to a single robustness value. Furthermore, the interval $(t_{s_i}, t_i]$ can have at most k_φ time-points. Thus, the computation of $[\rho](\varphi, \mathbf{x}_{[0,i]}, t_0)$ can be split into tracking a summary for the interval $[t_{s_i}, t_i]$ and tracking at most k_φ `RoSI`s in the worklists of the immediate subformulas of φ in the interval $(t_{s_i}, t_i]$. We now explain how the summary information is maintained for each formula.

(1) $[\Box\psi]$ We maintain the summary S = $\inf_{j \in [0,s_i]} [\rho](\psi, \mathbf{x}_{[0,i]}, t_j)$, i.e., the infimum over all exact robustness values computable over the partial signal $\mathbf{x}_{[0,i]}$. When a new time-point $(t_{i+1}, \mathbf{x}_{i+1})$ becomes available, S is updated if there is a new time $t_{s_{i+1}}$ for which $[\rho](\psi, \mathbf{x}_{[0,i]}, t_{s_{i+1}})$ can be exactly computed; otherwise, the new value is used to update all entries $[\rho](\psi, \mathbf{x}_{[0,i+1]}, t_j)$ for $t_j \in [t_{s_i+1}, t_i]$, and a new entry corresponding to time t_{i+1} is added to $\mathsf{worklist}[v_\psi]$. Please see Fig. 4 for a depiction of this step. We then establish the following: (1) There are at most k_φ entries (each corresponding to $[\rho](\psi, \mathbf{x}_{[0,i]}, t_j)$ for $t_j \in (t_{s_i}, t_i]$) in $\mathsf{worklist}[v_\psi]$. This is true because there can be at most k_φ consecutive time-points that do not update S in any interval of length w_φ. (2) We show by induction that the $\inf$ of S and the k_φ entries in $\mathsf{worklist}[v_\psi]$ is equal to $[\rho](\Box\psi, \mathbf{x}_{[0,i]}, t_0)$.

(2) $[\varphi\mathbf{U}\psi]$ We maintain the following two quantities as the summary:
(a) S_φ = $[\rho](\Box_{[0,t_{s_i}]}\varphi, \mathbf{x}_{[0,i]}, t_0)$ and (b) S_ψ = $[\rho](\varphi\mathbf{U}_{[0,t_{s_i}]}\psi, \mathbf{x}_{[0,i]}, t_0)$. In $\mathsf{worklist}[v_\varphi]$ and $\mathsf{worklist}[v_\psi]$ we store at most k_φ values corresponding to $[\rho](\varphi, \mathbf{x}_{[0,i]}, t_j)$ and $[\rho](\psi, \mathbf{x}_{[0,i]}, t_j)$ for $t_j \in (t_{s_i}, t_i]$. The crucial step is to combine S_φ and S_ψ with the entries in $\mathsf{worklist}[v_\varphi]$ and $\mathsf{worklist}[v_\psi]$ to obtain $[\rho](\varphi\mathbf{U}\psi, \mathbf{x}_{[0,i]}, t_0)$. We show that the iterative procedure in Algorithm 3 can accomplish this. In its j^{th} iteration v_1 is equal to $\inf\limits_{\ell \in [0,j]} [\rho](\varphi, \mathbf{x}_{[0,i]}, t_\ell)$, and we can show by induction that v_2 is equal to $\sup\limits_{m \in [0,j]} \min\left([\rho](\psi, \mathbf{x}_{[0,i]}, t_m), \inf\limits_{\ell \in [0,m]} [\rho](\varphi, \mathbf{x}_{[0,i]}, t_\ell)\right)$. Thus, at the end of the computation, the value computed in v_2 is $[\rho](\varphi\mathbf{U}\psi, \mathbf{x}_{[0,i]}, t_0)$.

(3) $[\Box\Diamond\psi]$ We show that we do not need any additional storage for monitoring φ. Concretely, we posit that $[\rho](\Box\Diamond\psi, \mathbf{x}_{[0,i]}, t_0)$ = $[\rho](\psi, \mathbf{x}_{[0,i]}, t_i)$. We successively rewrite $[\rho](\Box\Diamond\psi, \mathbf{x}_{[0,i]}, t_0)$ = $\inf\limits_{j \in [0,i]} \sup\limits_{\ell \in [j,i]} [\rho](\psi, \mathbf{x}_{[0,i]}, t_\ell)$ as follows:

Algorithm 4: Computing RoSI for $\Box(\varphi \vee \Diamond\psi)$

```
$v := \mathsf{S}$
foreach $j \in [s_i + 1, i]$ do
    $v := \sup([\rho](\psi, \mathbf{x}_{[0,i]}, t_j), \inf(v, [\rho](\varphi, \mathbf{x}_{[0,i]}, t_j)))$
$[\rho](\Box(\varphi \vee \Diamond\psi), \mathbf{x}_{[0,i]}, t_0) := v$
```

$$\inf\left(\sup_{\ell\in[0,i]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell),\ \sup_{\ell\in[1,i]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell),\ldots,\ \sup_{\ell\in[i,i]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell)\right) \tag{5.23}$$

$$\inf\begin{pmatrix}\sup([\rho](\psi,\mathbf{x}_{[0,i]},t_i), & \sup_{\ell\in[0,i-1]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell)) \\ \sup([\rho](\psi,\mathbf{x}_{[0,i]},t_i), & \sup_{\ell\in[1,i-1]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell)),\ldots,[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell)\end{pmatrix} \tag{5.24}$$

In the above, to go from (5.23) to (5.24), we expand the inner $\sup$ expressions, and observe that the last term in the $\inf$ evaluates to $[\rho](\psi, \mathbf{x}_{[0,i]}, t_i)$. For the final step, we observe that $\inf(I_1, \sup(I_1, I_2), \ldots, \sup(I_1, I_n)) = I_1$, and thus, (5.24) simplifies to $[\rho](\psi, \mathbf{x}_{[0,i]}, t_i)$ By duality, a similar proof works for $\Diamond\Box\psi$.

(4) $[\Box(\varphi\vee\Diamond\psi)]$ We maintain one quantity as the summary information: $\mathsf{S} = [\rho](\Box_{[0,s_i]}(\varphi\vee \Diamond\psi), \mathbf{x}_{[0,i]}, t_0)$. Additionally, we store at most k_φ entries corresponding to $[\rho](\varphi, \mathbf{x}_{[0,i]}, t_j)$ in $\mathsf{worklist}[v_\varphi]$ and at most k_φ entries corresponding to $[\rho](\psi, \mathbf{x}_{[0,i]}, t_j)$ in $\mathsf{worklist}[v_\psi]$. To compute $[\rho](\varphi, \mathbf{x}_{[0,i]}, t_0)$, we use Algorithm 4. To complete the proof we observe that Algorithm 4 computes expression (5.25) that has nested and alternating $\sup$s and $\inf$s:

$$\sup\left([\rho](\psi,\mathbf{x}_{[0,i]},t_i),\inf\left([\rho](\varphi,\mathbf{x}_{[0,i]},t_i),\sup\left([\rho](\psi,\mathbf{x}_{[0,i]},t_{i-1}),\ldots,\mathsf{S}\right)\right)\right) \tag{5.25}$$

Using the identity $\sup(I_1, \inf(I_2, I_3)) = \inf(\sup(I_1, I_2), \sup(I_1, I_3))$, we can rearrange the above expression to obtain:

$$\inf\begin{pmatrix}\sup\left([\rho](\psi,\mathbf{x}_{[0,i]},t_i),[\rho](\varphi,\mathbf{x}_{[0,i]},t_i)\right), \\ \sup\begin{pmatrix}[\rho](\psi,\mathbf{x}_{[0,i]},t_i),[\rho](\psi,\mathbf{x}_{[0,i]},t_{i-1}), \\ \inf\left([\rho](\varphi,\mathbf{x}_{[0,i]},t_{i-1}),\sup\left([\rho](\psi,\mathbf{x}_{[0,i]},t_{i-2}),\ldots,\mathsf{S}\right)\right)\end{pmatrix}\end{pmatrix} \tag{5.26}$$

By repeated use of this identity on the expression in the second line, we get the expression $\inf\limits_{j\in[0,i]}\left(\max\left([\rho](\varphi,\mathbf{x}_{[0,i]},t_j),\ \sup\limits_{\ell\in[j,i]}[\rho](\psi,\mathbf{x}_{[0,i]},t_\ell)\right)\right)$, which is equal to $[\rho](\varphi, \mathbf{x}_{[0,i]}, t_0)$.

■

6 Experimental Results

We implemented Algorithm 2 as a stand-alone tool that can be plugged in loop with any black-box simulator and evaluated it using two practical real-world applications. We considered the following criteria: (1) On an average, what fraction of simulation time can be saved by online monitoring? (2) How much overhead does online monitoring add, and how does it compare to a naïve implementation that at each step recomputes everything using an offline algorithm?

Requirement	Num.	Early	Simulation Time (hours)	
	Traces	Termination	Offline	Online
$\varphi_{overshoot}(\nu_1)$	1000	801	33.3803	26.1643
$\varphi_{overshoot}(\nu_2)$	1000	239	33.3805	30.5923
$\varphi_{overshoot}(\nu_3)$	1000	0	33.3808	33.4369
$\varphi_{transient}(\nu_4)$	1000	595	33.3822	27.0405
$\varphi_{transient}(\nu_5)$	1000	417	33.3823	30.6134

Table 1. Experimental results on DEM.

6.1 Diesel Engine Model (DEM)

The first case study is an industrial-sized Simulink®model of a prototype airpath system in a diesel engine. The closed-loop model consists of a plant model describing the airpath dynamics, and a controller implementing a proprietary control scheme. The model has more than 3000 blocks, with more than 20 lookup tables approximating high-dimensional nonlinear functions. Due to the significant model complexity, the speed of simulation is about 5 times slower, i.e., simulating 1 second of operation takes 5 seconds in Simulink®. As it is important to simulate this model over a long time-horizon to characterize the airpath behavior over extended periods of time, savings in simulation-time by early detection of requirement violations is very beneficial. We selected two parameterized safety requirements after discussions with the control designers, (shown in Eq. (6.1)-(6.2)). Due to proprietary concerns, we suppress the actual values of the parameters used in the requirements.

$$\varphi_{overshoot}(\mathbf{p_1}) = \Box_{[a,b]}(\mathbf{x} < c) \tag{6.1}$$

$$\varphi_{transient}(\mathbf{p_2}) = \Box_{[a,b]}(|\mathbf{x}| > c \implies (\Diamond_{[0,d]}|\mathbf{x}| < e)) \tag{6.2}$$

Property $\varphi_{overshoot}$ with parameters $\mathbf{p_1} = (a, b, c)$ specifies that in the interval $[a, b]$, the overshoot on the signal $\mathbf{x}$ should remain below a certain threshold c. Property $\varphi_{transient}$ with parameters $\mathbf{p_2} = (a, b, c, d, e)$ is a specification on the settling time of the signal $\mathbf{x}$. It specifies that in the time interval $[a, b]$ if at some time t, $|\mathbf{x}|$ exceeds c then it settles to a small region ($|\mathbf{x}| < e$) before $t + d$. In Table 1, we consider three different valuations ν_1, ν_2, ν_3 for $\mathbf{p_1}$ in the requirement $\varphi_{overshoot}(\mathbf{p_1})$, and two different valuations ν_4, ν_5 for $\mathbf{p_2}$ in the requirement $\varphi_{transient}(\mathbf{p_2})$.

The main reason for the better performance of the online algorithm is that simulations are time-consuming for this model. The online algorithm can terminate a simulation earlier (either because it detected a violation or obtained a concrete robust satisfaction interval), thus obtaining significant savings. For $\varphi_{overshoot}(\nu_3)$, we choose the parameter values for a and b such that the online algorithm has to process the entire signal trace, and is thus unable to terminate earlier. Here we see that the total overhead (in terms of runtime) incurred by the extra book-keeping by Algorithm 2 is negligible (about 0.1%).

6.2 CPSGrader

CPSGrader [14, 6] is a publicly-available automatic grading and feedback generation tool for online virtual labs in cyber-physical systems. It employs temporal logic based

STL Test Bench	Num.	Early	Sim. Time (mins)		Overhead (secs)	
	Traces	Termination	Offline	Online	Naïve	Alg. 2
avoid_front	1776	466	296	258	553	9
avoid_left	1778	471	296	246	1347	30
avoid_right	1778	583	296	226	1355	30
hill_climb$_1$	1777	19	395	394	919	11
hill_climb$_2$	1556	176	259	238	423	7
hill_climb$_3$	1556	124	259	248	397	7
filter	1451	78	242	236	336	6
keep_bump	1775	468	296	240	1.2×10^4	268
what_hill	1556	71	259	253	1.9×10^4	1.5×10^3

Table 2. Evaluation of online monitoring for CPSGrader. Each STL Test Bench has an associated STL property.

testers to check for common fault patterns in student solutions for lab assignments. CPSGrader uses the National Instruments Robotics Environment Simulator to generate traces from student solutions and monitors STL properties (each corresponding to a particular faulty behavior) on them. In the published version of CPSGrader [14], this is done in an offline fashion by first running the complete simulation until a pre-defined cut-off and then monitoring the STL properties on offline traces. At a step-size of 5 ms, simulating 6 sec. of real-world operation of the system takes 1 sec. for the simulator. When students use CPSGrader for active feedback generation and debugging, simulation constitutes the major chunk of the application response time. Online monitoring helps in reducing the response time by avoiding unnecessary simulations, giving the students feedback as soon as faulty behavior is detected.

We evaluated our online monitoring algorithm, on the traces and STL properties used in the published version of CPSGrader [14, 6]. These traces are the result of running actual student submissions on a battery of tests. For lack of space, we refer the reader to [14] for details about the tests and STL properties. As an illustrative example, we show the keep_bump property in Eq. 6.3:

$$\varphi_{\texttt{keep\_bump}} = \Diamond_{[0,60]} \Box_{[0,5]} \left(\texttt{bump\_right}(t) \vee \texttt{bump\_left}(t)\right) \tag{6.3}$$

For each STL property, Table 2 compares the total simulation time needed for both the online and offline approaches, summed over all traces. For the offline approach, a suitable simulation cut-off time of 60 sec. is chosen. At a step-size of 5 ms, each trace is roughly of length 1000. For the online algorithm, simulation terminates before this cut-off if the truth value of the property becomes known, otherwise it terminates at the cut-off. Table 2 also shows the monitoring overhead incurred by a naïve online algorithm that performs complete recomputation at every step against the overhead incurred by Alg. 2. Table 2 demonstrates that online monitoring ends up saving up to 24% simulation time ($>$ 10% in a majority of cases). The monitoring overhead of Alg. 2 is negligible ($<$ 1%) as compared to the simulation time and it is less than the overhead of the naïve online approach consistently by a factor of 40x to 80x.

References

1. Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *TACAS*, pages 254–257. 2011.
2. E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Formal Modeling and Analysis of Timed Systems*, pages 23–37. Springer International Publishing, 2014.
3. A. Dokhanchi, B. Hoxha, and G. Fainekos. On-line monitoring for temporal logic robustness. In *RV*, pages 231–246. 2014.
4. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, pages 167–170, 2010.
5. A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *CAV*, pages 264–279, 2013.
6. A. Donzé, G. Juniwal, J. C. Jensen, and S. A. Seshia. Cpsgrader website. `http://www.cpsgrader.org`.
7. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal modeling and analysis of timed systems*, pages 92–106. 2010.
8. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *CAV*, pages 27–39, 2003.
9. G. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using s-taliro. In *Proc. of the American Control Conference*, 2012.
10. G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comp. Sci.*, 410(42):4262–4291, 2009.
11. H.-M. Ho, J. Ouaknine, and J. Worrell. Online monitoring of metric temporal logic. In *Runtime Verification*. 2014.
12. B. Hoxha, H. Abbas, and G. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.
13. X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *Proc. of HSCC*, pages 43–52, 2013.
14. G. Juniwal, A. Donzé, J. C. Jensen, and S. A. Seshia. CPSGrader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory. In *EMSOFT*, October 2014.
15. Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta. Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 273–282. ACM, 2014.
16. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *arXiv preprint cs/0610046*, 2006.
17. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.
18. D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. *Formal Modeling and Analysis of Timed Systems*, pages 304–319, 2007.