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. Requirements of cyberphysical systems (CPS) can be rigorously specified using Signal Temporal Logic (STL). STL comes equipped with semantics that are able to quantify how robustly a given signal satisfies an STL property. In a setting where signal values over the entire time horizon of interest are available, efficient algorithms for *offline* computation of the robust satisfaction value have been proposed. Only a few methods exist for the *online* setting, i.e., where only a partial signal trace is available and rest of the signal becomes available in increments (such as in a real system or during numerical simulations). In this paper, we formalize the semantics for robust online monitoring of *partial signals* using the notion of robust satisfaction intervals (RoSIs). We propose an efficient algorithm to compute the RoSI and demonstrate its usage on two real-world case studies from the automotive domain and massively-online CPS education. As online algorithms permit early termination when the satisfaction or violation of a property is found, we show that savings in computationally expensive simulations far outweigh any overheads incurred by the online approach.

1 Introduction

Embedded software designers typically validate designs by inspecting concrete observations of system behavior. For instance, in the model-based development (MBD) paradigm, designers use numerical simulation tools to obtain traces from models of systems. An important problem is to efficiently test whether some logical property φ holds for a given simulation trace. It is increasingly common [15, 11, 14, 3, 18, 2, 16] to specify such properties using a real-time temporal logic such as Signal Temporal Logic (STL) [9] or Metric Temporal Logic (MTL) [12]. 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 T). Theoretical and practical results for offline monitoring [12, 7, 9, 20] 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 is impractical as it requires storing the entire observed trace. In a simulation tool that uses numerical techniques to compute system behaviors, obtaining even one signal trace

may require several minutes or even hours. If we wish to monitor a property over the simulation, it is usually sensible to stop the simulation once the satisfaction or violation of the property 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 [13, 10, 21]. Recent work shows 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, 6]. The quantitative semantics define a function mapping a property φ and a trace $\mathbf{x}(t)$ to a real number, known as the *robust satisfaction value*. A large positive value suggests that $\mathbf{x}(t)$ easily satisfies φ, a positive value near 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 [12, 9, 7], there is limited work on an online monitoring algorithm to do the same [5].

The theoretical challenge of online monitoring lies in the definition of a practical quantitative semantics for a temporal logic formula over a partial signal, i.e., a signal trace with incomplete data which may not yet validate or invalidate φ. Past work [10] has identified three views for the satisfaction of a LTL property φ over a partial trace τ: (1) a *weak view* where τ satisfies φ if there is some suffix τ' such that $\tau.\tau'$ satisfies φ, (2) a *strong view* where τ does not satisfy φ if there is some suffix τ' such that $\tau.\tau'$ does not satisfy φ and (3) a *neutral view* when the satisfaction is defined using a truncated semantics of LTL restricted to *finite* paths. In [13], the authors extend the truncated semantics to MTL. In [5], the authors introduce the notion of a *predictor*, which works as an oracle to complete a partial trace and provide an estimated satisfaction value. In general, such a value cannot be formally trusted as long as the data is incomplete.

The layout of the paper is as follows: In Section 3, we present *robust interval* semantics for an STL property φ on a partial signal (with data available till time t_i, denoted $\mathbf{x}_{[0,i]}$) that unifies the different semantic views of real-time logics on truncated paths. Informally, we define a function that maps $\mathbf{x}_{[0,i]}$ and φ to the robust satisfaction interval (`RoSI`) (ℓ, υ), with the interpretation that for any suffix $\mathbf{x}_{[t_{i+1},t_N]}$, ℓ is the greatest lower bound on the robust satisfaction value of $\mathbf{x}_N$, and υ is the corresponding lowest upper bound. There is a natural correspondence between the `RoSI` and three-valued semantics: (1) φ is violated according to the weak view iff υ is negative, and is satisfied otherwise; (2) φ is satisfied according to the strong view iff ℓ is positive, and violated otherwise; and (3) a neutral semantics, e.g., based on some predictor, can be defined when $\ell < 0 < \upsilon$, i.e., when there exist suffixes that can violate or satisfy φ.

compute the `RoSI` for a bounded-time-horizon formula. Our approach

In Section 4, we present an efficient online algorithm to compute the `RoSI` for a bounded-time-horizon STL formula by extending the offline algorithm of [7]. In spite of being online, the extension imposes minimal runtime overhead. It works in a fashion

similar to incremental Boolean monitoring of STL implemented in the tool AMT [21]. In Section 5, we present algorithms that can perform online monitoring of commonly-used unbounded time-horizon formulas using only a bounded amount of memory.

Finally, we present 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 [17]. Since the online algorithm can abort simulation as soon as the satisfaction 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.

$$\begin{array}{lll} 1.\ -I_1 = (-b_1, -a_1) & 3.\ I_1 \oplus I_2 = (a_1 + a_2, b_1 + b_2) & \\ 2.\ c + I_1 = (c + a_1, c + b_1) & 4.\ \min(I_1, I_2) = (\min(a_1, a_2), \min(b_1, b_2)) & \\ 5.\ I_1 \cap I_2 = \begin{cases} \emptyset & \text{if } \min(b_1, b_2) < \max(a_1, a_2) \\ (\max(a_1, a_2), \min(b_1, b_2)) & \text{otherwise.} \end{cases} & \end{array} \tag{2.1}$$

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* $\mathcal{T}' \subseteq \mathcal{T}$.

Note that $\mathcal{X}$ can be any set, but it is usual to assume some subset of $\mathbb{R}^n$. 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) [9] 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 [9]; we include the definition below. In the usual Boolean sense of satisfaction, a signal $\mathbf{x}$ satisfies φ at a time τ iff the robust satisfaction value $\rho(\varphi, \mathbf{x}, \tau) \geq 0$.

Definition 2 (Robust Satisfaction Value). *We first define a function ρ mapping an STL formula φ, the signal $\mathbf{x}$, and a time $\tau \in \mathcal{T}$ as follows:*

$$\begin{array}{lcl} \rho\left(f(\mathbf{x}) > 0, \mathbf{x}, \tau\right) & = & f(\mathbf{x}(\tau)) \\ \rho\left(\neg\varphi, \mathbf{x}, \tau\right) & = & -\rho(\varphi, \mathbf{x}, \tau) \\ \rho\left(\varphi_1 \wedge \varphi_2, \mathbf{x}, \tau\right) & = & \min\left(\rho(\varphi_1, \mathbf{x}, \tau), \rho(\varphi_2, \mathbf{x}, \tau)\right) \\ \rho\left(\Box_I \varphi, \mathbf{x}, \tau\right) & = & \inf_{t \in \tau + I} \rho(\varphi, \mathbf{x}, t) \\ \rho\left(\Diamond_I \varphi, \mathbf{x}, \tau\right) & = & \sup_{t \in \tau + I} \rho(\varphi, \mathbf{x}, t) \\ \rho\left(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}, \tau\right) & = & \sup\limits_{t_2 \in \tau + I} \min\left(\rho(\varphi_2, \mathbf{x}, t_2), \inf\limits_{t_1 \in (\tau, t_2)} \rho(\varphi_1, \mathbf{x}, t_1)\right) \end{array} \tag{2.3}$$

The robust satisfaction value *of a given signal $\mathbf{x}$ w.r.t. a given formula φ is then defined as $\rho(\varphi, \mathbf{x}, 0)$.*

3 Robust Interval Semantics

We assume finite time-horizon T for signals. Further, we assume that the signal is obtained by applying constant interpolation to a sampled signal defined over time-instants $\{t_0, t_1, \ldots, t_N\}$, such that $t_N = T$ and $\forall i : t_i < t_{i+1}$. In the 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 new semantics for STL formulas over partial signals using intervals. 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 `RoSI` of an STL formula with respect to a partial signal, and next 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 $t_i \leq T$, 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 $\tau \in [t_0, t_i]$ is an interval I s.t.:*

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

Definition 5. *We 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 $[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)$.*

$$
\begin{array}{lcl}
[\rho]\left(f(\mathbf{x}_{[0,i]}) > 0, \mathbf{x}_{[0,i]}, \tau\right) & = & \begin{cases} [f(\mathbf{x}_{[0,i]}(\tau)), f(\mathbf{x}_{[0,i]}(\tau))] \ \tau \in [t_0, t_i] \\ [f_{\mathsf{inf}}, f_{\mathsf{sup}}] \qquad\qquad\qquad\qquad \textit{otherwise.} \end{cases} \\
[\rho]\left(\neg\varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & -[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau) \\
[\rho]\left(\varphi_1 \wedge \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) & = & \min([\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau), [\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau)) \\
[\rho]\left(\Box_I \varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & \inf_{t \in \tau + I}\left([\rho](\varphi, \mathbf{x}_{[0,i]}, t)\right) \\
[\rho]\left(\Diamond_I \varphi, \mathbf{x}_{[0,i]}, \tau\right) & = & \sup_{t \in \tau + I}\left([\rho](\varphi, \mathbf{x}_{[0,i]}, t)\right) \\
[\rho]\left(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) & = & \sup\limits_{t_2 \in \tau + I} \min \left(\begin{array}{l} [\rho](\varphi_2, \mathbf{x}_{[0,i]}, t_2), \\ \inf\limits_{t_1 \in (\tau, t_2)} [\rho](\varphi_1, \mathbf{x}_{[0,i]}, t_1)) \end{array} \right)
\end{array}
\tag{3.1}
$$

It can be shown that the `RoSI` of a signal $\mathbf{x}$ w.r.t. an STL formula φ is equal to $[\rho](\varphi, \mathbf{x}, 0)$; we defer the proof to the full version [4].

4 Online Algorithm

Donzé et al. [7] 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 [7], an essential ingredient of the online algorithm is Lemire's running maximum filter algorithm [19]. The problem this algorithm addresses is the following: given a sequence of values $a_1, \ldots, a_n$, find the maxima over windows of size w, i.e., for all j, find $\max_{i \in [j, j+w)} a_i$ (similarly, for finding the corresponding minima). 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 (denoted `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). The descending edge satisfies the property that if $i \in \mathtt{F}$, then $t_i \in t + [a, b]$, and for all $t_j > t_i$ in $t + I$, $\mathbf{x}(t_j) < \mathbf{x}(t_i)$. Lines 8 and 9 incrementally update the edge when a new point is encountered that is still within the $t + [a, b]$ window, and lines 11,12,13 correspond to the case where the window is slid right as a result of updating the t. These lines then providing the sliding maximum over $t + [a, b]$ at the t from which the window was advanced.

We first focus on the fragment of STL where each temporal operator is scoped by a time-interval I, where $\sup(I)$ is finite. The algorithm for online monitoring maintains

Algorithm 1: SlidingMax($(t_0, \mathbf{x}_0), \ldots, (t_N, \mathbf{x}_N)$, $[a, b]$).

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 ≠ ∅ then t := min(t_{min(F)} − a, t_{i+1} − b)
    else t := t_{i+1} − b
    if t = t_{i+1} − b then
        while x_{i+1} ≥ x_{max(F)} ∧ F ≠ ∅ do
            F := F − max(F)
        F := F ∪ {i + 1}, i := i + 1
    else // Slide window to the right
        if s > t_0 then y(s) := x_{min(F)}
        else y(t_0) := x_{min(F)}
        F := F − min(F), s := t
```

the syntax tree of the formula φ to be monitored in memory, and augments the tree 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 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 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 `RoSI`s for its constituent sub-formulas in a bottom-up fashion. As computing the `RoSI` at a node v requires the `RoSI`s at the child-nodes, this computation has to be delayed till the `RoSI`s 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).

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

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. For the formula[4] in (4.2), we show $\mathcal{T}_\varphi$ and $\mathsf{hor}(v)$ for each v in $\mathcal{T}_\varphi$ in Fig. 1.

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

[3] We omit the case of $\mathbf{U}_I$ here for lack of space, although the rewriting approach of [7] can be adapted here and is 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.

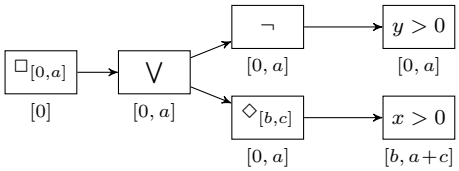

Fig. 1. Syntax tree $\mathcal{T}_\varphi$ for φ (given in (4.2)) with each node v annotated with $\mathsf{hor}(v)$.

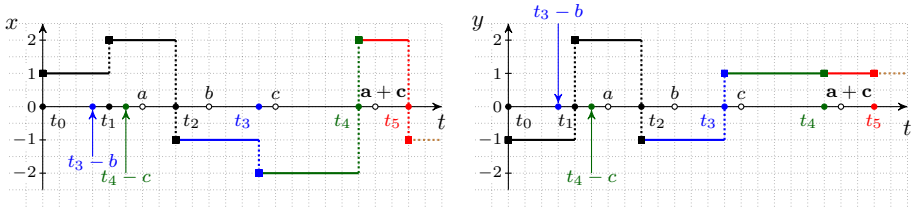

Fig. 2. These plots show the signals $x(t)$ and $y(t)$. Each signal begins at time $t_0 = 0$, and we consider three partial signals: $\mathbf{x}_{[0,3]}$ (black + blue), and $\mathbf{x}_{[0,4]}$ ($\mathbf{x}_{[0,3]}$ + green), and $\mathbf{x}_{[0,5]}$ ($\mathbf{x}_{[0,4]}$ + red).

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 $\texttt{RoSI}$s 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 max. 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 max. over the interval $[0+b, a+c]$, with a window of length $c-b$. We can use Algorithm 1 for this purpose. One caveat is that we need to store separate monotonic edges for the upper and

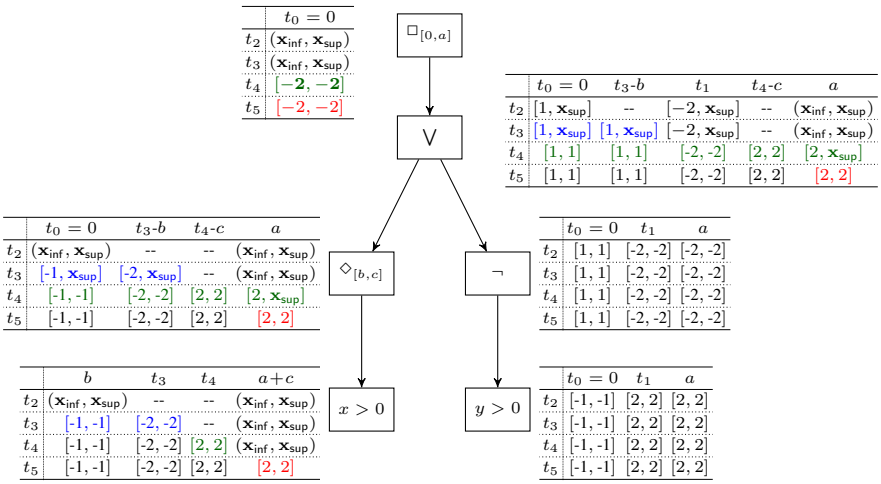

Fig. 3. We show a snapshot of the worklist$[v]$ maintained by the algorithm for four different (incremental) partial traces of the signals $x(t)$ and $y(t)$. Each row indicates the state of worklist$[v]$ at the time indicated in the first column. An entry marked -- indicates that the corresponding element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

lower bounds of the RoSIs. The algorithm then proceeds upward on the syntax tree, only updating the worklist of a node 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 worklist$[v_{y>0}]$ or worklist$[v_{\neg y>0}]$, as all RoSIs are already singular, but does update the RoSI values for the node $v_{x>0}$. The algorithm then invokes Alg. 1 on worklist$[v_{x>0}]$ to update 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 SlidingMax (worklist$[v_{\mathbf{x}>0}]$, $[b, c]$) $(t_3 - b)$, which is then inserted in 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 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 hor(φ).

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 predicate of the form $f(\mathbf{x}) > 0$. Here, the algorithm inserts the pair $(t_{i+1}, \mathbf{x}_{i+1})$ in worklist$[v_{f(\mathbf{x})>0}]$ if t_{i+1} lies inside hor$(v_{f(\mathbf{x})>0})$. In other words, it only tracks a value if it is useful for computing the RoSI 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 the current node represents $\neg\varphi$ (Line 7), the algorithm flips the sign of each entry in

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

```
   // v_ψ is a node in the syntax tree, (t_{i+1}, x_{i+1}) is a new timepoint
 1 switch ψ do
 2     case f(x) > 0 do
 3         if t_{i+1} ∈ hor(v_ψ) then
 4             worklist[v_ψ](t_{i+1}) := [f(x_{i+1}), f(x_{i+1})]
 5     case ¬φ do
 6         updateWorkList(v_φ, t_{i+1} ,x_{i+1}) ;
 7         worklist[v_ψ] := −worklist[v_φ]
 8     case φ_1 ∧ φ_2 do
 9         updateWorkList(v_{φ_1}, t_{i+1}, x_{i+1}) ;
10         updateWorkList(v_{φ_2}, t_{i+1}, x_{i+1}) ;
11         worklist[v_ψ] := min(worklist[v_{φ_1}], worklist[v_{φ_2}])
12     case □_I φ do
13         updateWorkList(v_φ, t_{i+1} , x_{i+1}) ;
14         worklist[v_ψ] := SlidingMax(worklist[v_φ], I)
```

worklist$[v_\varphi]$; this operation is denoted as $-$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 worklist$[v_{\varphi_1}]$ and worklist$[v_{\varphi_1}]$ can be each thought of as a piecewise-constant signal (likewise for 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 RoSIs of its arguments. Note that if for $i = 1, 2$, if 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, 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 worklist$[v_\varphi]$. It then applies Alg. 1 to compute the sliding maximum over worklist$[v_\varphi]$. Note that if worklist$[v_\varphi]$ contains N entries, so does worklist$[v_{\Diamond_I \varphi}]$.

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 a time horizon that is unbounded, or in other words, the algorithm would have to keep track of every value over arbitrarily long intervals. For a large

class of formulae (shown in Theorem 1), we can can perform robust online monitoring using only a constant amount of memory. The question whether an arbitrary STL formula outside of the fragment stated thus far can be monitored using constant memory remains an open problem. We now show how constant memory monitoring can be performed for the first set of formulae. In what follows, we assume that the subformulae φ and ψ are atomic predicates of the form $f(\mathbf{x}) > 0$. Also, we assume that the domain of signals is a compact set, and replace $\inf$ and $\sup$ by $\min$ and $\max$ respectively.

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. We consider each of the five cases of the theorem in turn. The proof strategy is to show that if a constant memory buffer has been used to monitor up to n samples, then receiving an additional sample does not require the memory to grow. 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) $\square(\varphi \vee \diamondsuit\psi)$, where $\varphi \equiv f(\mathbf{x}) > 0$, and $\psi \equiv g(\mathbf{x}) > 0$. Observe the following:

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

Repeatedly 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 $\diamondsuit(f(\mathbf{x}) > 0 \wedge \square(g(\mathbf{x}) > 0))$.

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

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

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 $\square\diamondsuit(f(\mathbf{x}) > 0)$, 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 $\diamondsuit\square(f(\mathbf{x}) > 0)$.

(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.

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

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

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

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

We defer the proof of the general case to the full version of the paper [4], but remark that the proof techniques are very similar. Let w denote the least upper bound of the

time horizon for all subformulae of a given untimed formula. At any time t_n, additional book-keeping is required to store partial information for time-points in the range $[t_n - w, t_n]$. By the step-size condition there can be at most $\lceil \frac{w}{\Delta} \rceil$ time-points in this range. This is then used to show that constant memory proportional to $\lceil \frac{w}{\Delta} \rceil$ is sufficient to monitor such an untimed formula (with bounded-horizon subformulae).

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?

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

Requirement	Num. Traces	Early Termination	Simulation Time (hours)	
			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.

STL Test Bench	Num. Traces	Early Termination	Sim. Time (mins)		Overhead (secs)	
			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
$\texttt{hill\_climb}_1$	1777	19	395	394	919	11
$\texttt{hill\_climb}_2$	1556	176	259	238	423	7
$\texttt{hill\_climb}_3$	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.

(in terms of runtime) incurred by the extra book-keeping by Algorithm 2 is negligible (about 0.1%).

CPSGrader.
CPSGrader [17, 8] is a publicly-available automatic grading and feedback generation tool for online virtual labs in cyber-physical systems. It employs temporal logic based 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 [17], 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 Alg. 2 on the signals and STL properties used in CPSGrader [17, 8]. These signal traces result from running actual student submissions on a battery of tests such as failure to avoid obstacles in front, failure to re-orient after obstacle avoidance, failure to reach the target region (top of a hill), failure to detect the hill, and failure to use a correct filter in order to climb a hill. For lack of space, we refer the reader to [17] for further details. As an illustrative example, consider `keep_bump` property in Eq. 6.3:

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

The `keep_bump` formula checks whether when the bump signal is activated (i.e., the robot bumps into an obstacle either from the left or the right), the controller keeps moving forward for some time instead of driving back in order to avoid the obstacle. 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.

7 Conclusions and Future Work

We have defined robust interval semantics for Signal Temporal Logic formulas over partial signal traces. The robust satisfaction interval (`RoSI`) of a partial signal contains the robust satisfaction value of any possible suffix of the given partial signal. We present an online algorithm to compute `RoSI` for a large class of STL formulas. Generalizations to full STL and considering signal traces defined by piecewise linear interpolation over given discrete-time points are important directions for future work.

Acknowledgments

This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, by NSF Expeditions grant CCF-1139138, and by Toyota under the CHESS center at UC Berkeley.

References

1. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In: Proc. of Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257 (2011)

2. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System Design of Stochastic Models using Robustness of Temporal Properties. Theor. Comp. Sci. (2015)
3. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-Driven Statistical Learning of Temporal Logic Properties. In: Proc. of Formal Modeling and Analysis of Timed Systems. pp. 23–37 (2014)
4. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust Online Monitoring of Signal Temporal Logic (2015), arXiv pre-print
5. Dokhanchi, A., Hoxha, B., Fainekos, G.: On-line Monitoring for Temporal Logic Robustness. In: Proc. of Runtime Verification. pp. 231–246 (2014)
6. Donzé, A.: Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In: Proc. of Computer-Aided Verification. pp. 167–170 (2010)
7. Donzé, A., Ferrère, T., Maler, O.: Efficient Robust Monitoring for STL. In: Proc. of Computer-Aided Verification. pp. 264–279 (2013)
8. Donzé, A., Juniwal, G., Jensen, J.C., Seshia, S.A.: CPSGrader website. `http://www.cpsgrader.org`
9. Donzé, A., Maler, O.: Robust Satisfaction of Temporal Logic over Real-valued Signals. In: Proc. of Formal Modeling and Analysis of Timed Systems. pp. 92–106 (2010)
10. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with Temporal Logic on Truncated Paths. In: Proc. of Computer-Aided Verification. pp. 27–39 (2003)
11. Fainekos, G., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of Automotive Control Applications using S-TaLiRo. In: Proc. of the American Control Conference (2012)
12. Fainekos, G.E., Pappas, G.J.: Robustness of Temporal Logic Specifications for Continuous-Time Signals. Theor. Comp. Sci. 410(42), 4262–4291 (2009)
13. Ho, H.M., Ouaknine, J., Worrell, J.: Online Monitoring of Metric Temporal Logic. In: Proc. of Runtime Verification (2014)
14. Hoxha, B., Abbas, H., Fainekos, G.: Benchmarks for temporal logic requirements for automotive systems. In: Proc. of Applied Verification for Continuous and Hybrid Systems (2014)
15. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining Requirements from closed-loop Control Models. In: Proc. of Hybrid Systems: Computation and Control. pp. 43–52 (2013)
16. Jones, A., Kong, Z., Belta, C.: Anomaly detection in cyber-physical systems: A formal methods approach. In: Proc. of IEEE Conf. on Decision and Control. pp. 848–853 (2014)
17. Juniwal, G., Donzé, A., Jensen, J.C., Seshia, S.A.: CPSGrader: Synthesizing Temporal Logic Testers for Auto-Grading an Embedded Systems Laboratory. In: Proc. of Conference on Embedded Software (October 2014)
18. Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., Belta, C.: Temporal Logic Inference for Classification and Prediction from Data. In: Proc. of Hybrid Systems: Computation and Control. pp. 273–282 (2014)
19. Lemire, D.: Streaming Maximum-Minimum Filter Using no More Than Three Comparisons per Element. arXiv preprint cs/0610046 (2006)
20. Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Proc. of Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166 (2004)
21. Nickovic, D., Maler, O.: AMT: A property-based monitoring tool for analog systems. In: Proc. of Formal Modeling and Analysis of Timed Systems. pp. 304–319 (2007)