Form Methods Syst Des (2017) 51:5–30
DOI 10.1007/s10703-017-0286-7

Robust online monitoring of signal temporal logic

Jyotirmoy V. Deshmukh[1] · Alexandre Donzé[2] · Shromona Ghosh[3] · Xiaoqing Jin[1] · Garvit Juniwal[3] · Sanjit A. Seshia[3]

Published online: 27 July 2017

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

A. Donzé did most of the work while being affiliated with the University of California, Berkeley.

✉ Jyotirmoy V. Deshmukh
jyotirmoy.deshmukh@toyota.com

✉ Alexandre Donzé
alex@decyphir.com

Shromona Ghosh
shromona.ghosh@eecs.berkeley.edu

Garvit Juniwal
garvitjuniwal@eecs.berkeley.edu

Sanjit A. Seshia
sseshia@eecs.berkeley.edu

[1] Toyota Motors North America R&D, Gardena, CA, USA

[2] Decyphir Inc., San Francisco, CA, USA

[3] University of California, Berkeley, CA, USA

setting. We show that savings in computationally expensive simulations far outweigh any overheads incurred by an online approach.

Keywords Cyberphysical systems · Runtime verification · Online monitoring · Signal temporal logic · Quantitative semantics

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 [2,9,12,15,17] 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 [5,7,10,19] 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 [8,11,21]. 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 [5,7,10], 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 Sect. 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 Sect. 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 [21]. 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 Sect. 5, we examine the case of *unbounded* horizon formulas, and present a proof that we can always design a robust online monitor using only a bounded amount of memory.

Finally, we present an implementation and illustrate it with a model of a fuel control system for gasoline engines proposed as industrial benchmark for the verification of Cyber-Physical Systems (CPS) in [14]. We then provide experimental results on two large-scale case studies: (i) industrial-scale Simulink models from the automotive domain in Sect. 6, and (ii) an automatic grading system used in a massive online education initiative on CPS [16]. 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

2.1 Interval arithmetic

We make extensive use of simple interval arithmetic operations which we review next. 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. In what follows, let $I_j = (a_j, b_j)$.

$$\begin{aligned} -I_1 &= (-b_1, -a_1) \\ c + I_1 &= (c + a_1, c + b_1) \end{aligned}$$

$$
\begin{aligned}
I_1 \oplus I_2 &= (a_1 + a_2, b_1 + b_2)\\
I_1 \cap I_2 &= \begin{cases} \emptyset & \text{if } \min(b_1, b_2) < \max(a_1, a_2)\\ (\max(a_1, a_2), \min(b_1, b_2)) & \text{otherwise.}\end{cases}\\
\min_j(I_j) &= (\min_j(a_j), \min_j(b_j))\\
\max_j(I_j) &= (\max_j(a_j), \max_j(b_j))
\end{aligned}
\tag{2.1}
$$

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 $\mathcal{X} \subseteq \mathbb{R}^m$ where m is the dimension of the signal. Given a time domain $\mathcal{T}$, a *partial signal* is any signal defined on a time domain $\mathcal{T}' \subseteq \mathcal{T}$

We remark that we consider the signal-value domain $\mathcal{X}$ to be some compact (bounded and closed) subset of $\mathbb{R}^m$. This is so because we assume that signals are obtained from simulations of physical systems or sensor measurements from a real-world system. In either case, it is reasonable to assume that the signal values come from a bounded set, as physical quantities (e.g., temperature, pressure, velocity) are either inherently bounded, or are limited to the resolution offered by the measurement systems.

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.

2.2 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 $\mathbb{R}$. The syntax of an STL formula φ is defined as follows:

$$
\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi \mathbf{U}_I \varphi \mid \varphi \mathbf{R}_I \varphi \tag{2.2}
$$

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:

$$
\begin{aligned}
\rho\,(f(\mathbf{x}){>}0, \mathbf{x}, \tau) &= f(\mathbf{x}(\tau))\\
\rho\,(\neg\varphi, \mathbf{x}, \tau) &= -\rho(\varphi, \mathbf{x}, \tau)\\
\rho\,(\varphi_1 \wedge \varphi_2, \mathbf{x}, \tau) &= \min\left(\rho(\varphi_1, \mathbf{x}, \tau), \rho(\varphi_2, \mathbf{x}, \tau)\right)\\
\rho\,(\Box_I \varphi, \mathbf{x}, \tau) &= \inf_{\tau' \in \tau + I} \rho(\varphi, \mathbf{x}, \tau')\\
\rho\,(\Diamond_I \varphi, \mathbf{x}, \tau) &= \sup_{\tau' \in \tau + I} \rho(\varphi, \mathbf{x}, \tau')\\
\rho\,(\varphi \mathbf{U}_I \psi, \mathbf{x}, \tau) &= \sup_{\tau_1 \in \tau + I} \min\left(\rho(\psi, \mathbf{x}, \tau_1), \inf_{\tau_2 \in [\tau, \tau_1]} \rho(\varphi, \mathbf{x}, \tau_2)\right)
\end{aligned}
$$

$$\rho\left(\varphi \mathbf{R}_I \psi, \mathbf{x}, \tau\right) = \inf_{\tau_1 \in \tau + I} \max\left(\rho(\psi, \mathbf{x}, \tau_1), \sup_{\tau_2 \in [\tau, \tau_1]} \rho(\varphi, \mathbf{x}, \tau_2)\right) \tag{2.3}$$

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 $\rho(\varphi, \mathbf{x}, \tau) \geq 0$. We say a signal $\mathbf{x}$ satisfies φ iff $\rho(\varphi, \mathbf{x}) = \rho(\varphi, \mathbf{x}, 0) \geq 0$, i.e., it satisfies φ at time 0.

Remark 1 The semantics presented here and in [7,19] have a technical difference from the ones used in the conventional definition used for Linear Temporal Logic and Metric Temporal Logic (such as in [11]). The difference lies in the definition of $\mathbf{U}$, where for the formula $\varphi \mathbf{U} \psi$, we require φ to hold at the time-point at which ψ holds, while previous approaches do require it. (Similarly for the $\mathbf{R}$ operator).

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 $\{\mathbf{x} \mid \mathbf{x}_{[0,i]} \text{is a prefix of} \, \mathbf{x}\}$.

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:

$$\begin{aligned} \inf(I) &= \inf_{\mathbf{x} \in \mathcal{C}(\mathbf{x}_{[0,i]})} \rho(\varphi, \mathbf{x}, \tau) \\ \sup(I) &= \sup_{\mathbf{x} \in \mathcal{C}(\mathbf{x}_{[0,i]})} \rho(\varphi, \mathbf{x}, \tau) \end{aligned} \tag{3.1}$$

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 $[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau)$. We use the notation f_{inf} and f_{sup} to respectively denote the infimal and supremal value of the function $f(\mathbf{x})$ over the signal domain $\mathcal{X}$.

$$\begin{aligned} [\rho]\left(f(\mathbf{x}_{[0,i]}) {\geq} 0, \mathbf{x}_{[0,i]}, \tau\right) &= \begin{cases} [f(\mathbf{x}_{[0,i]}(\tau)), f(\mathbf{x}_{[0,i]}(\tau))] & \tau \in [t_0, t_i] \\ [f_{\mathsf{inf}}, f_{\mathsf{sup}}] & \text{otherwise.} \end{cases} \\ [\rho]\left(\neg\varphi, \mathbf{x}_{[0,i]}, \tau\right) &= -[\rho](\varphi, \mathbf{x}_{[0,i]}, \tau) \\ [\rho]\left(\varphi_1 \wedge \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) &= \min([\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau), [\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau)) \end{aligned}$$

$$[\rho]\left(\Box_I\varphi, \mathbf{x}_{[0,i]}, \tau\right) = \inf_{\tau'\in\tau+I}\left([\rho](\varphi, \mathbf{x}_{[0,i]}, \tau')\right)$$

$$[\rho]\left(\Diamond_I\varphi, \mathbf{x}_{[0,i]}, \tau\right) = \sup_{\tau'\in\tau+I}\left([\rho](\varphi, \mathbf{x}_{[0,i]}, \tau')\right)$$

$$[\rho]\left(\varphi_1\mathbf{U}_I\varphi_2, \mathbf{x}_{[0,i]}, \tau\right) = \sup_{\tau_1\in\tau+I}\min\begin{pmatrix}[\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau_1), \\ \inf\limits_{\tau_2\in[\tau,\tau_1]}[\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau_2)\end{pmatrix}$$

$$[\rho]\left(\varphi_1\mathbf{R}_I\varphi_2, \mathbf{x}_{[0,i]}, \tau\right) = \inf_{\tau_1\in\tau+I}\max\begin{pmatrix}[\rho](\varphi_2, \mathbf{x}_{[0,i]}, \tau_1), \\ \sup\limits_{\tau_2\in[\tau,\tau_1]}[\rho](\varphi_1, \mathbf{x}_{[0,i]}, \tau_2)\end{pmatrix} \tag{3.2}$$

The following lemma shows that the interval obtained by applying the recursive definition for $[\rho]$ is indeed the robust satisfaction interval as defined in Definition 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.*

Proof We prove by induction on the formula structure. The base case is when φ is μ. If $\tau \in [t_0, t_i]$ then $[\rho](\varphi, \mathbf{y}, \tau) = [f(\mathbf{y}(\tau)), f(\mathbf{y}(\tau))]$. Let $\mathbf{x}$ be any completion of $\mathbf{y}$. If $\tau \in [t_0, t_i]$ then $\mathbf{x}(\tau) = \mathbf{y}(\tau)$, and hence, $\inf_{\mathbf{x}\in\mathcal{C}(\mathbf{y})}\rho(\varphi, \mathbf{x}, \tau)$ (resp. sup) is equal to $f(\mathbf{y}(\tau))$. If $\tau > t_i$, then $[\rho](\varphi, \mathbf{y}, \tau) = [f_{\mathsf{inf}}, f_{\mathsf{sup}}]$. By definition of f_{inf} and f_{sup}, for any completion $\mathbf{x}$, $f(\mathbf{x}(\tau)) \in [f_{\mathsf{inf}}, f_{\mathsf{sup}}]$.

We proceed by structural induction on the formula structure. Assume $[\rho](\varphi, \mathbf{y}, \tau)$ is a RoSI, i.e., for any completion $\mathbf{x}$ of $\mathbf{y}$, $\rho(\varphi, \mathbf{x}, \tau) \in [\rho](\varphi, \mathbf{y}, \tau)$. As $\rho(\neg\varphi, \mathbf{x}, \tau) = -\rho(\varphi, \mathbf{x}, \tau)$, it follows that if $\rho(\varphi, \mathbf{x}, \tau) \in [\rho](\varphi, \mathbf{y}, \tau)$, then $-\rho(\varphi, \mathbf{x}, \tau) \in -[\rho](\varphi, \mathbf{y}, \tau)$, by interval arithmetic. For all the remaining operators, we can make a similar argument, and the results follow by the properties of interval arithmetic. □

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 a basic online algorithm for a bounded-horizon fragment of STL.

As in the offline monitoring algorithm in [5], an essential ingredient of the online algorithm is Lemire's running maximum filter algorithm [18]. The problem this algorithm addresses is the following: given a sequence of values $a_1, \ldots, a_n$, find the maximum over all windows of size w, i.e., for all j, $\max_{i\in[j,j+w)} a_i$. 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 monotonic edge (noted `F` in Algorithm 1) to compute the sliding maxima, in order to achieve an optimal procedure, measured in terms of the number of comparisons between elements. The first element of `F` at time t is the sliding maximum at time t. As F slides forward in time and the window hits a new value $\mathbf{x}_{i+1}$ (Line 6 in Algorithm 1), F is updated by removing every element indexed by the tail of F that are smaller than $\mathbf{x}_{i+1}$ (Line 7–8), then $i+1$ becomes the tail. It can be shown that with this mechanism, the total number of comparisons between

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

elements is linear in the total number of values. Algorithm 1 extends [18] in that it deals with timed values (or piecewise constant signals) instead of a simple discrete array and is slightly simpler than the one in [5] which considers piecewise linear signals, i.e., linear interpolation between values.

4.1 Stipulation on time-step

As in the offline algorithm, we stipulate that the signal being monitored has finite variability. We formalize this in terms of a minimum time-step in the signal. Given a signal $\mathbf{x}(t)$ as a sequence of time-value pairs $(t_0, \mathbf{x}_0), \ldots, (t_n, \mathbf{x}_n)$, we require n to be a finite integer, and we require that there exist a known $\Delta \in \mathbb{R}^{>0}$, such for any $i \geq 0$, $t_{i+1} - t_i > \Delta$.

We observe that the knowledge of such a "minimum time-step" Δ known *a priori* is a reasonable assumption for most real-time monitoring scenarios. This is so because, it is physically impossible for a monitoring routine to samples the actual dense-time signal at an infinite resolution, i.e., some time must pass between consecutive samples in the signal.

We now present the algorithm for 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 an STL operator $\neg, \vee, \wedge, \Box_I$ or $\Diamond_I$. We omit the case of $\mathbf{U}_I$ here for clarity of the presentation - simply note that the rewriting approach of [5] can also be adapted and was implemented in our tool. 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 sub-formulas 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).

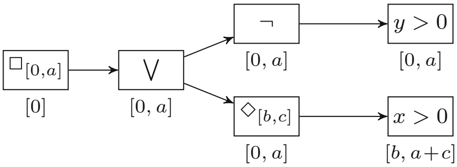


Fig. 1 Syntax tree $\mathcal{T}_\varphi$ for φ [given in (4.2)] with each node v annotated with $\mathsf{hor}(v)$

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

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 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.[1]

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

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 $[\rho](\psi, \mathbf{x}_{[0,i+1]}, t)$(Fig. 2).

In Fig. 3, we give an example of a run of the algorithm on the plots shown in Fig. 2. 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 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 $\texttt{RoSI}$s. 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

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

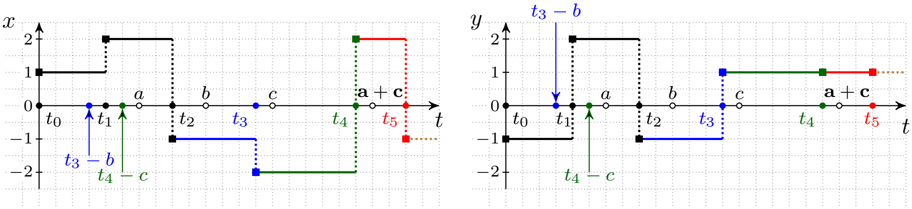
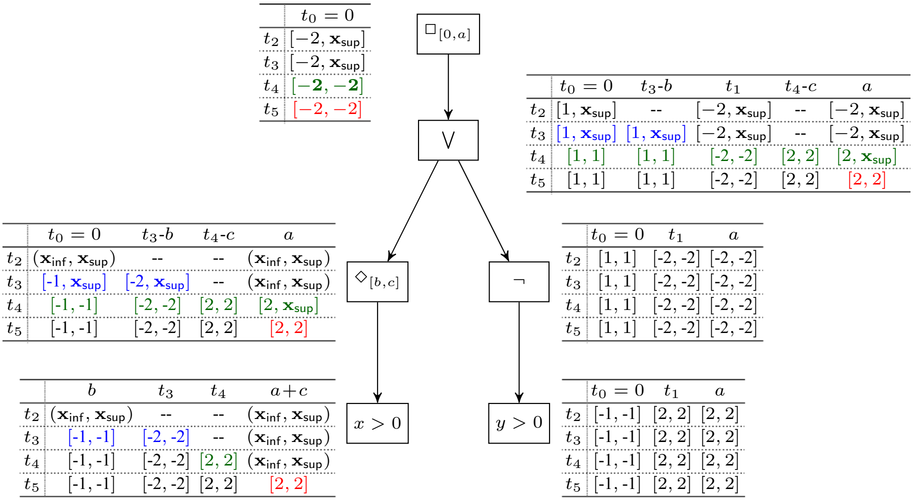


Fig. 2 These plots show the signals $x(t)$ and $y(t)$. Each signal begins at time $t_0 = 0$, and we consider three partial signals: $\mathbf{x}_{[0,3]}$ (*black* + *blue*), and $\mathbf{x}_{[0,4]}$ ($\mathbf{x}_{[0,3]}$ + *green*), and $\mathbf{x}_{[0,5]}$ ($\mathbf{x}_{[0,4]}$ + *red*). (Color figure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



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

	$t_0 = 0$	t_3-b	t_1	t_4-c	a
t_2	$[1, \mathbf{x}_{\mathsf{sup}}]$	--	$[-2, \mathbf{x}_{\mathsf{sup}}]$	--	$[-2, \mathbf{x}_{\mathsf{sup}}]$
t_3	$[1, \mathbf{x}_{\mathsf{sup}}]$	$[1, \mathbf{x}_{\mathsf{sup}}]$	$[-2, \mathbf{x}_{\mathsf{sup}}]$	--	$[-2, \mathbf{x}_{\mathsf{sup}}]$
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*. (Color figure online)

RoSIs are already singular, but does update the RoSI values for the node $v_{x>0}$. The algorithm then invokes Algorithm 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 Algorithm 1 to produce a new value of $\mathsf{SlidingMax}\,(\mathsf{worklist}[v_{\mathbf{x}>0}], [b, c])\,(t_3 - b)$, 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 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 $\mathsf{hor}(\varphi)$.

Algorithm 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

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
1  switch $\psi$ do
2      case $f(\mathbf{x}) > 0$
3          if $t_{i+1} \in \mathsf{hor}(v_\psi)$ then
4              worklist[$v_\psi$]($t_{i+1}$) := $[f(\mathbf{x}_{i+1}), f(\mathbf{x}_{i+1})]$
5      case $\neg\varphi$
6          updateWorkList($v_\varphi$, $t_{i+1}$, $\mathbf{x}_{i+1}$) ;
7          worklist[$v_\psi$] := $-$worklist[$v_\varphi$]
8      case $\varphi_1 \wedge \varphi_2$
9          updateWorkList($v_{\varphi_1}$, $t_{i+1}$, $\mathbf{x}_{i+1}$) ;
10         updateWorkList($v_{\varphi_2}$, $t_{i+1}$, $\mathbf{x}_{i+1}$) ;
11         worklist[$v_\psi$] := min(worklist[$v_{\varphi_1}$], worklist[$v_{\varphi_2}$])
12     case $\square_I \varphi$
13         updateWorkList($v_\varphi$, $t_{i+1}$, $\mathbf{x}_{i+1}$) ;
14         worklist[$v_\psi$] := $-$SlidingMax($-$worklist[$v_\varphi$], $I$)
15     case $\Diamond_I \varphi$
16         updateWorkList($v_\varphi$, $t_{i+1}$, $\mathbf{x}_{i+1}$) ;
17         worklist[$v_\psi$] := SlidingMax(worklist[$v_\varphi$], $I$)
```

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 the current node represents $\neg\varphi$ (Line 5), the algorithm flips the sign of each entry in worklist[v_φ]; this operation is denoted as $-$worklist[v_φ]. 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_{φ_1}] and worklist[v_{φ_1}] can be each thought of as a piecewise-constant signal (likewise for worklist[v_{φ_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 for $i = 1, 2$, if worklist[v_{φ_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., $\Diamond_I \varphi$ or $\square_I \varphi$, the algorithm first updates worklist[v_φ]. It then applies Algorithm 1 to compute the sliding maximum over worklist[v_φ] for $\Diamond_I$. For $\square_I$, the expression $-$SlidingMax($-$worklist[v_φ], I) effectively computes the sliding minimum by reusing the SlidingMax algorithm. Note that if worklist[v_φ] contains N entries, so does worklist[$v_{\Diamond_I \varphi}$] or worklist[$v_{\square_I \varphi}$]. We note that the minimum time-step stipulation enforces that all the worklists have a finite number of elements.

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_φ] are available. The optimization is to only compute SlidingMax over worklist[v_φ] starting from $u + \inf(I)$. We omit the pseudo-code for brevity. Finally, we remark that the robust satisfaction interval shrinks (or stays the same) with every new time-point in the signal that becomes available, and never grows.

5 Nominal semantics and monitoring untimed formulas

In the previous section, we presented a monitoring algorithm for STL formulas with a bounded time-horizon. In this section, we consider STL formulas that may have temporal operators with an unbounded time-horizon. We first make the observation that the robust interval semantics for a large class of unbounded-horizon STL formulas such as $\Box\Diamond(f(\mathbf{x}) > 0)$, or $\Box(\varphi \implies \Diamond\psi)$ is trivially $[f_{\mathsf{inf}}, f_{\mathsf{sup}}]$.

Clearly, considering robust interval semantics for unbounded horizon STL formulas is thus trivial and not useful, as it does not give any information. In this section, we define *nominal* robust semantics for the purpose of online monitoring.

5.1 Nominal semantics

The nominal semantics of an STL formula can be described in terms of the recursive function ρ_{nom} that maps a given formula φ, a partial signal $\mathbf{x}_{[0,i]}$ and a time $\tau \in \mathcal{T}$ to a value $\rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,i]}, \tau)$.

$$\begin{aligned}
\rho_{\text{nom}}\left(f(\mathbf{x}_{[0,i]}) > 0, \mathbf{x}_{[0,i]}, \tau\right) &= f(\mathbf{x}_{[0,i]}(\tau))\\
\rho_{\text{nom}}\left(\neg\varphi\right) &= -\rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,i]}, \tau)\\
\rho_{\text{nom}}\left(\varphi_1 \wedge \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) &= \min(\rho_{\text{nom}}(\varphi_1, \mathbf{x}_{[0,i]}, \tau), \rho_{\text{nom}}(\varphi_2, \mathbf{x}_{[0,i]}, \tau))\\
\rho_{\text{nom}}\left(\Box_I\varphi, \mathbf{x}_{[0,i]}, \tau\right) &= \inf_{\tau' \in (\tau + I \cap [0,t_i])} \left(\rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,i]}, \tau')\right)\\
\rho_{\text{nom}}\left(\Diamond_I\varphi, \mathbf{x}_{[0,i]}, \tau\right) &= \sup_{\tau' \in (\tau + I \cap [0,t_i])} \left(\rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,i]}, \tau')\right)\\
\rho_{\text{nom}}\left(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) &= \sup_{\tau_2 \in (\tau + I \cap [0,t_i])} \min\begin{pmatrix}\rho_{\text{nom}}(\varphi_2, \mathbf{x}_{[0,i]}, \tau_2),\\ \inf\limits_{\tau_1 \in [\tau,\tau_2]} \rho_{\text{nom}}(\varphi_1, \mathbf{x}_{[0,i]}, \tau_1)\end{pmatrix}\\
\rho_{\text{nom}}\left(\varphi_1 \mathbf{R}_I \varphi_2, \mathbf{x}_{[0,i]}, \tau\right) &= \inf_{\tau_2 \in (\tau + I \cap [0,t_i])} \max\begin{pmatrix}\rho_{\text{nom}}(\varphi_2, \mathbf{x}_{[0,i]}, \tau_2),\\ \sup\limits_{\tau_1 \in [\tau,\tau_2]} \rho_{\text{nom}}(\varphi_1, \mathbf{x}_{[0,i]}, \tau_1)\end{pmatrix} \qquad (5.1)
\end{aligned}$$

Observe that the nominal robust satisfaction value closely matches the usual definition for the robust satisfaction value, except that for the unbounded temporal operators, the robustness computation is restricted to the signal values available at time t_i.

Note that the algorithm to compute robust interval satisfaction could be used to compute the nominal robust satisfaction, but a direct application of Algorithm 2 requires every node in the sub-tree rooted at the untimed operator to have a time horizon that is equal to the time horizon for the trace. In other words, for all such nodes, the algorithm would have to keep track of every value over arbitrarily long intervals. In what follows, we show that we can monitor arbitrary STL formulas using amount of memory that is trace-length independent, such that the bound is exponential in the size of the formula. We first establish this result for a fragment of STL (denoted STL_u) that does not contain any bounded time-horizon temporal operators, and then indicate how it can be generalized to entire STL.

In what follows, we assume that the domain of signals is a compact set, and replace inf and sup by min and max respectively. The syntax of a formula φ in STL_u is as defined in (5.2). Here, μ indicates a predicate over signals, same as the one defined for STL in (2.2).

Note that in STL$_u$ we assume that all negations are pushed to the μ atoms using standard rewrite rules for STL.

$$\varphi ::= \mu \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \varphi \mathbf{U} \varphi \varphi \mathbf{R} \varphi \tag{5.2}$$

Proposition 1 *For $t_i \in [t_0, t_{n-1}]$, we can always rewrite the nominal robustness of a STL$_u$ formula as an expression of the following form:*

$$\rho_{\mathrm{nom}}(\phi, \mathbf{x}_{[0,n]}, t_i) = \max \begin{pmatrix} \alpha(n), \\ \min\left(\beta_1(n), \rho_{\mathrm{nom}}(\zeta_1, \mathbf{x}_{[0,n-1]}, t_i)\right) \\ \vdots \\ \min\left(\beta_k(n), \rho_{\mathrm{nom}}(\zeta_k, \mathbf{x}_{[0,n-1]}, t_i)\right) \end{pmatrix} \tag{5.3}$$

We prove this proposition using structural induction on STL$_u$ formulas. We use the following expression as an abbrevication for the expression in Eq. (5.3):

$$E\left(t_i, \alpha(n), (\beta_1(n), \zeta_1), \ldots, (\beta_k(n), \zeta_k)\right).$$

Note that the values $\alpha(n)$ or $\beta_\ell(n)$, $\ell \in [1, k]$, could be $-\infty$ or $+\infty$. Further, if a $\beta_\ell(n)$ value is $-\infty$, then this term can be dropped from the expression, and if an $\alpha(n)$ value is $+\infty$, then the whole expression evaluates to $+\infty$ (possibly indicating that the formula being monitored is a tautology).

5.2 Intuition for computing memory bounds

In the above expression, $\alpha(n)$ and $\beta_j(n)$ are meant to represent concrete numeric values evaluate that are computed from the formula to be monitored using trace-values only for time t_n. The ζ_j formulas are some STL$_u$ formulas related to the original formula. This relation will become clearer from the proofs of the lemmata to follow. Each of the expression $\rho_{\mathrm{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_i)$ represents a variable that stores the nominal robustness of the formula ζ_ℓ over the partial signal $\mathbf{x}_{[0,n-1]}$. In other words, each such formula represents a *summary* of the computation till time t_{n-1}. Thus, for any given formula, the number of the ζ terms in the expression to compute the nominal robustness of that formula determines how many variables required. Thus, if the number of ζ terms is finite, i.e., dependent only on the length of the formula, and *independent* of the trace-length, then we can monitor the robust nominal robustness over unbounded horizon temporal formulas with finite memory.

5.3 Intuition for the inductive hypothesis

The main intuition for the inductive hypothesis is that for any STL$_u$ formula, the expression for the nominal robustness can always be rewritten as a "sum of products" form, i.e., as a max of (1) a specific numeric value obtained only from the signal value at time t_n, and (2) expressions representing the min of a *summary variable* representing computation over the signal values from time t_0 to t_{n-1} with a specific numeric value computed only using the data at time t_n.

We now prove the validity of the proposition by structural induction over the formula structure, i.e., assuming that the hypothesis holds for subformulas, we prove that the hypothesis holds for the formula constructed using the subformulas. We start with the base case where the STL$_u$ formula is an atomic proposition, and show that the induction hypothesis is valid for this case.

Lemma 2 (Base Case) *The nominal robustness for a formula μ, where $\mu \equiv (f(\mathbf{x}) > 0)$ at time $t_i \in [t_0, t_{n-1}]$ over the partial signal $\mathbf{x}_{[0,n]}$ can be written as an expression of the form in Eq.* (5.3).

Proof As the nominal robustness for μ at time $t_i \in [t_0, t_{n-1}]$ is independent of $\mathbf{x}(t_n)$, we can see that $\rho_{\text{nom}}(\mu, \mathbf{x}_{[0,n]}, t_i)$ is equivalent to $\rho_{\text{nom}}(\mu, \mathbf{x}_{[0,n-1]}, t_i)$, which can be written as $\max(-\infty, \min(\infty, \rho_{\text{nom}}(\mu, \mathbf{x}_{[0,n-1]}, t_i)))$, i.e. $E\,(t_i, -\infty, (+\infty, \mu))$. □

We now establish Proposition 1 inductively for other kinds of STL$_u$ formulas.

Lemma 3 *The nominal robustness of $\varphi_1 \wedge \varphi_2$ at time $t_i \in [t_0, t_{n-1}]$ over the partial signal $\mathbf{x}_{[0,n]}$ can be written as an expression of the form* (5.3).

Proof By definition of nominal robustness,

$$\rho_{\text{nom}}(\varphi_1 \wedge \varphi_2, \mathbf{x}_{[0,n]}, t_i) = \min(\rho_{\text{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_i), \rho_{\text{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_i)). \tag{5.4}$$

Using the induction hypothesis, we can replace each argument of the min as follows:

$$\min\left(\begin{array}{l} \max\left(\begin{array}{l} \alpha^1(n), \\ \min\left(\beta^1_1(n), \rho_{\text{nom}}\left(\zeta^1_1, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \dots \\ \min\left(\beta^1_{k_1}(n), \rho_{\text{nom}}\left(\zeta^1_{k_1}, \mathbf{x}_{[0,n-1]}, t_i\right)\right) \end{array}\right), \\ \max\left(\begin{array}{l} \alpha^2(n), \\ \min\left(\beta^2_1(n), \rho_{\text{nom}}\left(\zeta^2_1, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \dots, \\ \min\left(\beta^2_{k_2}(n), \rho_{\text{nom}}\left(\zeta^2_{k_2}, \mathbf{x}_{[0,n-1]}, t_i\right)\right) \end{array}\right) \end{array}\right) \tag{5.5}$$

Using the rule $\min(\max(a, b), \max(c, d)) = \max(\min(a, c), \min(a, d), \min(b, c), \min(b, d))$, we can rewrite the above formula as follows:

$$\max\left(\begin{array}{l} \min\left(\alpha^1(n), \alpha^2(n)\right), \\ \min\limits_{\ell \in [1,k_1]}\left(\alpha^1(n), \beta^2_\ell(n), \rho_{\text{nom}}\left(\zeta^2_\ell, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \min\limits_{m \in [1,k_2]}\left(\alpha^2(n), \beta^1_m(n), \rho_{\text{nom}}\left(\zeta^1_m, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \min\limits_{\ell \in [1,k_1]} \min\limits_{m \in [1,k_2]}\left(\min\left(\beta^1_\ell, \beta^2_m, \rho_{\text{nom}}\left(\zeta^1_\ell \wedge \zeta^2_m, \mathbf{x}_{[0,n-1]}, t_i\right)\right)\right) \end{array}\right) \tag{5.6}$$

Observe that the above formula is of the following form that preserves the induction hypothesis:

$$E\left(t_i, \begin{array}{lll} \min\left(\alpha^1(n), \alpha^2(n)\right), & & \\ \left(\min\left(\alpha^1(n), \beta^2_1(n)\right),\ \zeta^2_1\right), & \dots, & \left(\min\left(\alpha^1(n), \beta^2_{k_2}(n)\right),\ \zeta^2_{k_2}\right) \\ \left(\min\left(\alpha^2(n), \beta^1_1(n)\right),\ \zeta^2_1\right), & \dots, & \left(\min\left(\alpha^2(n), \beta^1_{k_1}(n)\right),\ \zeta^2_{k_1}\right), \\ \left(\min\left(\beta^1_1(n), \beta^2_1(n)\right),\ \zeta^1_1 \wedge \zeta^2_1\right)) & \dots & \left(\min\left(\beta^1_1(n), \beta^2_{k_2}(n)\right),\ \zeta^1_1 \wedge \zeta^2_{k_2}\right), \\ \vdots & \ddots & \vdots \\ \left(\min\left(\beta^1_{k_1}(n), \beta^2_1(n)\right),\ \zeta^1_{k_1} \wedge \zeta^2_1\right), & \dots, & \left(\min\left(\beta^1_{k_1}(n), \beta^2_{k_2}(n)\right),\ \zeta^1_{k_1} \wedge \zeta^2_{k_2}\right) \end{array}\right) \tag{5.7}$$

We observe that if the expressions for φ_1 and φ_2 respectively have $k_1 + 1$ and $k_2 + 1$ terms, then the expression for $\varphi_1 \wedge \varphi_2$ has $(k_1 + 1)(k_2 + 1)$ terms. □

Lemma 4 *The nominal robustness of $\varphi_1 \vee \varphi_2$ at time $t_i \in [t_0, t_{n-1}]$ over the partial signal $\mathbf{x}_{[0,n]}$ can be written as an expression of the form* (5.3).

Proof By definition of nominal robustness,

$$\rho_{\mathrm{nom}}(\varphi_1 \vee \varphi_2, \mathbf{x}_{[0,n]}, t_i) = \max(\rho_{\mathrm{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_i), \rho_{\mathrm{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_i)). \tag{5.8}$$

Using the induction hypothesis, we can replace each argument of the min as follows:

$$\max\left(\begin{array}{l} \max\left(\begin{array}{l} \alpha^1(n), \\ \min\left(\beta_1^1(n), \rho_{\mathrm{nom}}\left(\zeta_1^1, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \dots \\ \min\left(\beta_{k_1}^1(n), \rho_{\mathrm{nom}}\left(\zeta_{k_1}^1, \mathbf{x}_{[0,n-1]}, t_i\right)\right) \end{array}\right), \\ \max\left(\begin{array}{l} \alpha^2(n), \\ \min\left(\beta_1^2(n), \rho_{\mathrm{nom}}\left(\zeta_1^2, \mathbf{x}_{[0,n-1]}, t_i\right)\right), \\ \dots, \\ \min\left(\beta_{k_2}^2(n), \rho_{\mathrm{nom}}\left(\zeta_{k_2}^2, \mathbf{x}_{[0,n-1]}, t_i\right)\right) \end{array}\right) \end{array}\right) \tag{5.9}$$

Observe that the above formula is of the following form that preserves the induction hypothesis:

$$E\left(t_i, \begin{array}{l} \quad \max\left(\alpha^1(n), \alpha^2(n)\right), \\ \left(\beta_1^1(n),\ \zeta_1^1\right), \dots, \left(\beta_{k_1}^1(n),\ \zeta_{k_1}^1\right), \\ \left(\beta_1^2(n),\ \zeta_1^2\right), \dots, \left(\beta_{k_2}^2(n),\ \zeta_{k_2}^2\right) \end{array}\right) \tag{5.10}$$

Observe that if φ_1 and φ_2 have k_1 and k_2 summary variables respectively, $\varphi_1 \vee \varphi_2$ requires $k_1 + k_2$ summary variables. □

Lemma 5 *The nominal robustness of $\Diamond\varphi$ at time $t_i \in [t_0, t_{n-1}]$ over the partial signal $\mathbf{x}_{[0,n]}$ can be written as an expression of the form* (5.3).

Proof By definition of nominal robustness of $\Diamond\varphi$,

$$\begin{aligned} \rho_{\mathrm{nom}}(\Diamond\varphi, \mathbf{x}_{[0,n]}, t_i) &= \max_{t_j \in [t_i, t_n]} \rho_{\mathrm{nom}}(\varphi, \mathbf{x}_{[0,n]}, t_j) \\ &= \max\left(\varphi(n), \max_{t_j \in [t_i, t_{n-1}]} \rho_{\mathrm{nom}}(\varphi, \mathbf{x}_{[0,n-1]}, t_j)\right) \end{aligned} \tag{5.11}$$

The above is obtained by splitting the outer max into a term over time points between t_i and t_{n-1} and a second term containing the evaluation of the nominal robustness of the formula φ at time t_n. Note that for any formula φ, the nominal robustness at time t_n is a concrete value that can be computed using the recursive semantics. We call this value $\varphi(n)$. We now use the inductive hypothesis (Eq. (5.3)) for $\rho_{\mathrm{nom}}(\varphi, \mathbf{x}_{[0,n]}, t_j)$:

$$\max\left(\varphi(n), \max_{t_j \in [t_i, t_{n-1}]}\left(\alpha(n), \max_{\ell \in [1,k]} \min\left(\beta_\ell(n), \rho_{\mathrm{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_j)\right)\right)\right) \tag{5.12}$$

Grouping the $\alpha(n)$ term,

$$= \max\left(\alpha(n), \varphi(n), \max_{t_j \in [t_i, t_n]} \max_{\ell \in [1,k]} \left(\min\left(\beta_1(n), \rho_{\mathrm{nom}}(\zeta_1, \mathbf{x}_{[0,n-1]}, t_j)\right)\right)\right) \tag{5.13}$$

Distributing the inner max over individual time points $t_j \in [t_i, t_n]$:

$$= \max\left(\alpha(n), \varphi(n), \max\left(\begin{array}{l}\max\limits_{\ell\in[1,k]} \min\left(\beta_\ell(n), \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_i)\right),\\ \max\limits_{\ell\in[1,k]} \min\left(\beta_\ell(n), \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_{i+1})\right),\\ \ldots,\\ \max\limits_{\ell\in[1,k]} \min\left(\beta_\ell(n), \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_{n-1})\right)\end{array}\right)\right). \tag{5.14}$$

Observe that for each $\ell \in [1, k]$, we can use the following rule to group expressions containing $\beta_\ell(n)$ terms: $\max(\min(a, b), \min(a, c)) = \min(a, \max(b, c))$. Furthermore, based on the definition of $\Diamond$, we can have that $\max_{t_j\in[t_i,t_{n-1}]} \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_j)$ is equivalent to $\rho_{\text{nom}}(\Diamond\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_i)$.

$$= \max\left(\begin{array}{l}\max(\alpha(n), \varphi(n))\\ \min(\beta_1(n), \rho_{\text{nom}}(\Diamond\zeta_1, \mathbf{x}_{[0,n-1]}, t_i)),\\ \ldots,\\ \min(\beta_k(n), \rho_{\text{nom}}(\Diamond\zeta_k, \mathbf{x}_{[0,n-1]}, t_i))\end{array}\right) \tag{5.15}$$

The above expression is of the following form, which matches the inductive hypothesis specified in Eq. (5.3): $E\left(t_i, \max(\varphi(n), \alpha(n)), (\beta_1(n),\ \Diamond\zeta_1), \ldots, (\beta_k,\ \Diamond\zeta_k)\right)$. Furthermore, observe that if φ requires k summary variables, $\Diamond\varphi$ also requires k summary variables. □

Lemma 6 *The nominal robustness of* $\Box\varphi$ *at time* $t_i \in [t_0, t_{n-1}]$ *for the partial signal* $\mathbf{x}_{[0,n]}$ *can be written as an expression of the form* (5.3).

Proof By definition of nominal robustness of $\Box\varphi$,

$$\begin{aligned}\rho_{\text{nom}}(\Box\varphi, \mathbf{x}_{[0,n]}, t_i) &= \min_{t_j\in[t_i,t_n]} \rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,n]}, t_j)\\ &= \min\left(\varphi(n), \min_{t_j\in[t_i,t_{n-1}]} \rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,n]}, t_j)\right)\end{aligned} \tag{5.16}$$

As before, we use the inductive hypothesis for $\rho_{\text{nom}}(\varphi, \mathbf{x}_{[0,n]}, t_j)$:

$$= \min\left(\varphi(n), \min_{t_j\in[t_i,t_{n-1}]} \max\left(\alpha(n), \max_{\ell\in[1,k]} \min\left(\beta_\ell(n), \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_j)\right)\right)\right) \tag{5.17}$$

Grouping the $\alpha(n)$ terms:

$$= \min(\varphi(n), \max\left(\alpha(n), \underline{\min_{t_j\in[t_i,t_n]} \max_{\ell\in[1,k]} \min(\beta_\ell(n), \rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_j))}\right) \tag{5.18}$$

Let ζ_ℓ^j be the shorthand for $\rho_{\text{nom}}(\zeta_\ell, \mathbf{x}_{[0,n-1]}, t_j)$. We can rewrite the underlined term in Eq. (5.18) as follows:

$$\min_{t_j\in[t_i,t_{n-1}]}\left(\begin{array}{l}\max\left(\min\left(\beta_1, \zeta_1^i\right), \ldots, \min\left(\beta_k, \zeta_k^i\right)\right),\\ \max\left(\min\left(\beta_1, \zeta_1^{i+1}\right), \ldots, \min\left(\beta_k, \zeta_k^{i+1}\right)\right),\\ \ldots\\ \max\left(\min\left(\beta_1, \zeta_1^{n-1}\right), \ldots, \min\left(\beta_k, \zeta_k^{n-1}\right)\right)\end{array}\right) \tag{5.19}$$

Since min and max are distributive over each other, we can now repeatedly apply the rule $\min(\max(a, b), \max(c, d)) = \max(\min(a, c), \min(a, d), \min(b, c), \min(b, d))$ to the above expression. Observe that this will lead to an expression representing a max over an exponential number of terms. What will these terms look like? There will be $\binom{k}{1}$ terms of the form $\min(\beta_\ell, \zeta_\ell^i, \zeta_\ell^{i+1}, \zeta_\ell^{n-1})$. This is equivalent to $\min(\beta_\ell, \rho_{\text{nom}}(\Box(\zeta_\ell), \mathbf{x}_{[0,n-1]}, t_i))$. Then, there will be $\binom{k}{2}$ terms of the form:

$$\min\left(\beta_\ell, \beta_m, \max_{A \in \Pi_{j=1}^{n-1}\{\zeta_\ell^j, \zeta_m^j\}} \min A\right).$$

In the above expression the set A is an element of the $n-i$-fold Cartesian product of $\{\zeta_\ell^j, \zeta_m^j\}$. Note that there are 2^{n-i} elements in the Cartesian product. The reader can verify that this expression is actually equivalent to $\min(\beta_\ell, \beta_m, \min_{j \in [i,n-1]} \max(\zeta_\ell^j, \zeta_m^j))$. This in turn simplifies to $\min(\beta_\ell, \beta_m, \rho_{\text{nom}}(\Box(\zeta_\ell \vee \zeta_m), \mathbf{x}_{[0,n-1]}, t_i))$. We can continue this process considering terms that group three β values together, four β values, and so on. We can show that this results in expression (5.20) below. In this expression, K denotes $[1, k]$ (i.e., the set $\{1, \ldots, k\}$). For convenience, let $\binom{K}{\ell}$ denote the set of all ℓ element combinations of the set K. E.g., if $k = 3$, i.e., $K = \{1, 2, 3\}$, $\binom{K}{2} = \{(1, 2), (2, 3), (3, 1)\}$.

$$\max_{m \in K} \max_{L \in \binom{K}{m}} \min\left(\left(\min_{\ell \in L} \beta_\ell\right), \rho_{\text{nom}}\left(\Box\left(\bigvee_{\ell \in L} \zeta_\ell\right), \mathbf{x}_{[0,n-1]}, t_i\right)\right) \tag{5.20}$$

We can now substitute expression (5.20) in Eq. (5.18) and using the rule $\min(a, \max(b, c))$ = $\max(\min(a, b), \min(a, c))$, we get an expression that is of the form specified in Eq. (5.3):

$$E\left(\begin{array}{l} t_i, \min(\varphi(n), \alpha(n)), \\ \left(\begin{array}{l} \min_{\ell \in L} \min(\varphi(n), \beta_\ell(n)), \\ \Box\left(\bigvee_{\ell \in L} \zeta_\ell\right) \end{array}\right)_{\forall L \in \binom{K}{1}}, \ldots, \left(\begin{array}{l} \min_{\ell \in L} \min(\varphi(n), \beta_\ell(n)), \\ \Box\left(\bigvee_{\ell \in L} \zeta_\ell\right) \end{array}\right)_{\forall L \in \binom{K}{k}} \end{array}\right). \tag{5.21}$$

The total number of terms in the above expression is $\sum_{m=1}^{k} \binom{k}{m}$, i.e., 2^k. Thus, if the expression for φ has k summary variables, the expression for $\Box\varphi$ can require up to 2^k summary variables. □

Example 2 As the formula for $\Box\varphi$ is hard to parse, consider the following example for the expression corresponding to (5.20) for the case when $k = 3$, i.e., when the exression representing the nominal robustness of φ has three (β, ζ) pairs:

$$\max\left(\begin{array}{l} \min(\beta_1(n), \rho_{\text{nom}}(\Box\zeta_1, \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\beta_2(n), \rho_{\text{nom}}(\Box\zeta_2, \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\beta_3(n), \rho_{\text{nom}}(\Box\zeta_3, \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\min(\beta_1(n), \beta_2(n)), \rho_{\text{nom}}(\Box(\zeta_1 \vee \zeta_2), \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\min(\beta_1(n), \beta_3(n)), \rho_{\text{nom}}(\Box(\zeta_1 \vee \zeta_3), \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\min(\beta_2(n), \beta_3(n)), \rho_{\text{nom}}(\Box(\zeta_2 \vee \zeta_3), \mathbf{x}_{[0,n-1]}, t_i)), \\ \min(\min(\beta_1(n), \beta_2(n), \beta_3(n)), \rho_{\text{nom}}(\Box(\zeta_1 \vee \zeta_2 \vee \zeta_3), \mathbf{x}_{[0,n-1]}, t_i)) \end{array}\right) \tag{5.22}$$

Lemma 7 *The nominal robustness of* $\varphi\mathbf{U}\varphi_2$ *at time* $t_i \in [t_0, t_{n-1}]$ *for the partial signal* $\mathbf{x}_{[0,n]}$ *can be written as an expression of the form* (5.3).

Proof By the definition of nominal robustness:

$$\rho_{\text{nom}}(\varphi_1\mathbf{U}\varphi_2, \mathbf{x}_{[0,n]}, t_i) = \max_{t_j \in [t_i, t_n]} \min\left(\begin{array}{l} \min_{t_h \in [t_i, t_j]} \rho_{\text{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_h), \\ \rho_{\text{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_j) \end{array}\right) \tag{5.23}$$

Separating the computation for time t_n in the outermost max operation:

$$= \max \left(\begin{array}{l} \min \left(\varphi_2(n), \varphi_1(n), \underline{\min\limits_{t_h \in [t_i, t_{n-1}]} \rho_{\mathrm{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_h)} \right), \\ \max\limits_{t_j \in [t_i, t_{n-1}]} \min \left(\begin{array}{l} \min\limits_{t_h \in [t_i, t_j]} \rho_{\mathrm{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_h), \\ \rho_{\mathrm{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_j) \end{array} \right) \end{array} \right) \tag{5.24}$$

Observe that the underlined term in the above expression, is equivalent to $\rho_{\mathrm{nom}}(\Box\varphi_1, \mathbf{x}_{[0,n-1]}, t_i)$. We now analyze term in the second line of the above expression, i.e.,

$$\max_{t_j \in [t_i, t_{n-1}]} \min \left(\begin{array}{l} \rho_{\mathrm{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_j), \\ \min\limits_{t_h \in [t_i, t_j]} \rho_{\mathrm{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_h) \end{array} \right). \tag{5.25}$$

From the inductive hypothesis, we can write $\rho_{\mathrm{nom}}(\varphi_1, \mathbf{x}_{[0,n]}, t_h) = \max(\alpha^1(n), A_h)$, where:

$$A_h = \max_{\ell \in [1, k_1]} \min \left(\beta^1_\ell(n), \rho_{\mathrm{nom}} \left(\zeta^1_\ell, \mathbf{x}_{[0,n-1]}, t_h \right) \right)$$

Similarly, we can write $\rho_{\mathrm{nom}}(\varphi_2, \mathbf{x}_{[0,n]}, t_j) = \max(\alpha^2(n), B_j)$. Then, expression (5.25) can be written as:

$$\max \left(\begin{array}{l} \min \left(\max \left(\alpha^2(n), B_i \right), \max \left(\alpha^1(n), A_i \right) \right), \\ \min \left(\max \left(\alpha^2(n), B_{i+1} \right), \max \left(\alpha^1(n), A_i \right), \max \left(\alpha^1(n), A_{i+1} \right) \right), \\ \min \left(\max \left(\alpha^2(n), B_{n-1} \right), \min \left(\max \left(\alpha^1(n), A_i \right), \ldots, \max \left(\alpha^1(n), A_{n-1} \right) \right) \right) \end{array} \right) \tag{5.26}$$

We can rewrite the hth row of expression (5.26) as follows:

$$\min \left(\max \left(\alpha^2(n), B_j \right), \max \left(\alpha^1(n), \min_{h \in [i,j]} A_h \right) \right). \tag{5.27}$$

We use the equivalence $\min\limits_{h \in [i,j]} \max \left(\alpha^1, A_h \right) = \max \left(\alpha^1, \min_{j \in [i,n]} A_h \right)$ for the above rewriting. Distributing the min operation over the max operations in expression (5.27), we get:

$$\max \left(\begin{array}{ll} \min \left(\alpha^2(n), \alpha^1(n) \right), & \min \left(\alpha^2(n), \min\limits_{h \in [i,j]} A_h \right), \\ \min \left(B_j, \alpha^1(n) \right), & \min \left(B_j, \min\limits_{h \in [i,j]} A_h \right) \end{array} \right). \tag{5.28}$$

Substituting the above in Eq. (5.26), and grouping similar terms together, we get:

$$\max \left(\begin{array}{ll} \min \left(\alpha^2(n), \alpha^1(n) \right), & \underline{\min \left(\alpha^2(n), \max\limits_{j \in [i,n-1]} \min\limits_{h \in [i,j]} A_h \right),} \\ \underset{\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim}{\min \left(\alpha^1(n), \max\limits_{j \in [i,n-1]} B_j \right),} & \underset{-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-}{\max\limits_{j \in [i,n-1]} \min \left(B_j, \min\limits_{h \in [i,j]} A_h \right).} \end{array} \right) \tag{5.29}$$

The underlined expression above is equivalent to A_i. The expression with the squiggly underline reduces to the following expression:

$$\max_{m \in [1,k_2]} \left(\min \left(\alpha^1(n), \beta^2_m(n), \rho_{\mathrm{nom}} \left(\Diamond\zeta^2_m, \mathbf{x}_{[0,n-1]}, t_i \right) \right) \right). \tag{5.30}$$

We are then left with the term that has the dashed underline. Substituting the expressions for B_j and A_h, we get:

$$\max_{j\in[i,n-1]} \min \begin{pmatrix} \max\limits_{m\in[1,k_2]} \min\left(\beta_m^2(n),\, \zeta_m^{2,j}\right), \\ \min\limits_{h\in[i,j]}\ \max\limits_{\ell\in[1,k_1]} \min\left(\beta_\ell^1(n),\, \zeta_\ell^{1,h}\right) \end{pmatrix} \tag{5.31}$$

We can expand this formula over the outermost j index, and start grouping terms in a fashion similar to the technique in the proof of Lemma (6). We can show that we get an expression of the following form:

$$\max \begin{pmatrix} \max\limits_{\substack{m\in[1,k_2]\\ \ell\in[1,k_1]}} \min\left(\beta_m^2(n),\, \beta_\ell^1(n),\, \rho_{\text{nom}}\left(\zeta_\ell^1 \mathbf{U} \zeta_m^2,\, \mathbf{x}_{[0,n-1]},\, t_i\right)\right), \\ \max\limits_{\substack{m\in[1,k_2]\\ \ell_1\in[1,k_1]\\ \ell_2\in[1,k_1]}} \min\left(\beta_m^2(n),\, \beta_{\ell_1}^1(n),\, \beta_{\ell_2}^1(n),\, \rho_{\text{nom}}\left(\left(\zeta_{\ell_1}^1 \vee \zeta_{\ell_2}^1\right) \mathbf{U} \zeta_m^2,\, \mathbf{x}_{[0,n-1]},\, t_i\right)\right), \\ \ldots,\ldots, \\ \max\limits_{m\in[1,k_2]} \min\left(\beta_m^2(n),\, \min\limits_{\ell\in[1,k_1]} \beta_\ell^1(n),\, \rho_{\text{nom}}\left(\left(\bigvee\limits_{\ell\in[1,k_1]} \zeta_\ell^1\right) \mathbf{U} \zeta_m^2,\, \mathbf{x}_{[0,n-1]}\right)\right) \end{pmatrix} \tag{5.32}$$

We can now substitute expressions (5.30) and (5.32) in expression (5.29), and then substitute the second term in Eq. (5.24) with the resulting expression, we get the following expression for the nominal robustness for $\varphi_1 \mathbf{U} \varphi_2$:

$$E \begin{pmatrix} t_i,\, \min\left(\alpha^1(n),\, \alpha^2(n)\right), \\ (\varphi_2(n),\ \Box\varphi_1), \\ \left(\min\left(\beta_m^2(n),\, \alpha^1(n)\right),\ \Diamond\zeta_m^2\right)_{m\in[1,k_2]}, \\ \left(\min\left(\beta_m^2(n),\, \min_{\ell\in L} \beta_\ell^1(n)\right),\ \left(\bigvee_{\ell\in L} \zeta_\ell^1\right) \mathbf{U} \zeta_m^2\right)_{\{L\in\binom{K_1}{r} \mid r\in K_1\},\, m\in[1,k_2]} \end{pmatrix} \tag{5.33}$$

Observe that if φ_1 and φ_2 have respectively k_1 and k_2 summary variables, then the nominal robustness for $\varphi_1 \mathbf{U} \varphi_2$ needs $k_2 + k_2 \cdot 2^{k_1}$ summary variables. □

Lemma 8 *The nominal robustness of $\varphi \mathbf{R} \varphi_2$ at time $t_i \in [t_0, t_{n-1}]$ for the partial signal $\mathbf{x}_{[0,n]}$ can be written as an expression of the form (5.3).*

Proof The proof for the $\mathbf{R}$ operator is very similar to that of the $\mathbf{U}$ and the $\Box$ operator. For brevity, we omit the tedious steps required to show that the expression for nominal robustness for $\varphi_1 \mathbf{R} \varphi_2$ can be written as:

$$E \begin{pmatrix} t_i,\, \min\left(\varphi_2(n),\, \alpha^2(n)\right), \\ \left(\min\left(\varphi_2(n),\, \min\limits_{m\in M} \beta_m^2(n)\right),\ \Box\left(\bigvee_{m\in M} \zeta_m^2\right)\right), \\ \left(\min\limits_{m\in M} \beta_m^2(n),\ \Diamond\varphi_1 \wedge \Box\left(\bigvee\limits_{m\in M} \zeta_m^2\right)\right), \\ \left(\min\left(\min\limits_{m\in M} \beta_\ell^2(n),\, \min\limits_{\ell\in L} \beta_\ell^1(n),\, \varphi_2(n)\right),\ \left(\bigvee_{\ell\in L} \zeta_\ell^1\right) \mathbf{R} \left(\bigvee_{m\in M} \zeta_m^2\right)\right), \\ \left(\min\left(\min\limits_{m\in M} \beta_\ell^2(n),\, \min\limits_{\ell\in L} \beta_\ell^1(n)\right),\ \Diamond\varphi_1 \wedge \left(\bigvee_{\ell\in L} \zeta_\ell^1\right) \mathbf{R} \left(\bigvee_{m\in M} \zeta_m^2\right)\right), \\ (\varphi_2(n),\ \varphi_1),\, (+\infty,\ \Diamond\varphi_1) \end{pmatrix} \tag{5.34}$$

In the above expression, for the second and third lines, M is an element of $\binom{K_2}{r}$, for every $r \in K_2$. For the fourth and fifth lines, M is an element of $\binom{K_2}{r}$, for every r in K_2, and L is an element of $\binom{K_1}{r}$ for every r in K_1. Observe that if φ_1 and φ_2 have respectively k_1 and k_2 summary variables, then the nominal robustness for $\varphi_1 \mathbf{R} \varphi_2$ requires $2^{k_1+k_2} + 2^{k_2} + 1$ summary variables. □

Theorem 1 *The robust satisfaction interval for any formula φ belonging to the syntactic fragment STL$_u$ can be monitored in an online fashion using a fixed amount of memory that depends on a function of $|\varphi|$ where $|\varphi|$ indicates the length of φ, and independent of the trace length.*

Proof The proof follows from the proofs of Lemmata 2, 3, 4, 5, 6, 7 and 8. Substituting $t_i = 0$ in the expression for nominal robustness of the top-level formula. Note that any subformula with top-level operators $\mathbf{U}$ or $\Box$ requires memory that is exponential in the number of variables required to monitor its subformulae. However, the memory required is independent of trace-length, and dependent only on the size of the formula. □

We now give a couple of examples to show how the lemmata in the above proof can be applied to obtain an algorithmic procedure to monitor STL$_u$ formulae.

Example 3 Consider the formula $\Box\Diamond(x > 0)$. Applying Lemma 2, the nominal robustness for $x > 0$ is:

$$E\,(t_i, -\infty, (+\infty, x > 0)) \tag{5.35}$$

From Lemma 5, the nominal robustness for $\Diamond(x > 0)$ is:

$$\begin{aligned} &E\,(t_i, \max(-\infty, x(n)), (+\infty, \Diamond(x > 0))) \\ &\quad = E\,(t_i, x(n), \min(+\infty, \Diamond(x > 0))) \end{aligned} \tag{5.36}$$

From Lemma 6, we can write the nominal robustness for $\Box\Diamond(x > 0)$:

$$E\begin{pmatrix} t_i, \max(x(n), x(n)), \\ (\min(x(n), +\infty),\ \Box\Diamond(x > 0)) \end{pmatrix} \tag{5.37}$$

We can see that above expression simplies to simply $x(n)$. In other words, nominal robustness of $\Box\Diamond(x > 0)$ depends only on the value of $x(t)$ at time t_n.

Example 4 Consider the formula $\Box\,((x > 0) \vee \Diamond(y > 0))$. Applying Lemma 2, the nominal robustness for the formula $(y > 0)$ is the following expression:

$$E\,(t_i, -\infty, (+\infty, y > 0)) \tag{5.38}$$

From Lemma 5, we can write the nominal robustness for formula $\Diamond(y > 0)$ as follows:

$$\begin{aligned} &E\,(t_i, \max(-\infty, y(n)), (+\infty, \Diamond(y > 0))) \\ &\quad = E\,(t_i, y(n), \min(+\infty, \Diamond(y > 0))) \end{aligned} \tag{5.39}$$

Similar to (5.38), the nominal robustness for $(x > 0)$ is $E\,(t_i, -\infty, (+\infty, x)))$. From Lemma 4, we can write the nominal robustness for $(x > 0) \vee \Diamond(y > 0)$ as:

$$\begin{aligned} &E\,(t_i, \max(-\infty, y(n)), (+\infty, x > 0), (+\infty, \Diamond(y > 0))) \\ &\quad = E\,(t_i, y(n), (+\infty, x > 0), (+\infty, \Diamond(y > 0))) \end{aligned} \tag{5.40}$$

Finally, using Lemma (6), the nominal robustness for $\square((x > 0) \vee \Diamond(y > 0))$ at time 0 can be written as:

$$E\begin{pmatrix} 0, \min(\max(x(n), y(n)), y(n)), \\ (\min(+\infty, \max(x(n), y(n))),\ \square(x > 0))\,, \\ (\min(+\infty, \max(x(n), y(n))),\ \square(\Diamond(y > 0)))\,, \\ (\min(+\infty, \max(x(n), y(n))),\ \square(x > 0 \vee \Diamond(y > 0))) \end{pmatrix} \tag{5.41}$$

From Example 3, we can see that $\rho_{\text{nom}}(\square(\Diamond(y > 0)), \mathbf{x}_{[0,n-1]}, 0)$ is $y(n-1)$, so the above expression simplifies to:

$$\max\begin{pmatrix} y(n), \min(\max(x(n), y(n)), \rho_{\text{nom}}(\square(x > 0), \mathbf{x}_{[0,n-1]}, 0)), \\ \min(\max(x(n), y(n)), y(n-1)), \\ \min(\max(x(n), y(n)), \rho_{\text{nom}}(\square(x > 0 \vee \Diamond(y > 0)), \mathbf{x}_{[0,n-1]}, 0)) \end{pmatrix} \tag{5.42}$$

This shows that we can monitor the nominal robustness of $\square(x > 0 \vee \Diamond(y > 0)))$ using two variables (one for $\square(x > 0)$ and one for $\square(x > 0 \vee \Diamond(y > 0)))$. In the case of this formula, we can show that this can be further simplified to:

$$\max(y(n), \min(x(n), \rho_{\text{nom}}(\square(x > 0 \vee \Diamond(y > 0)), \mathbf{x}_{[0,n-1]}, 0))). \tag{5.43}$$

This allows us to monitor the formula with just one variable.

Example 5 Consider the formula $\Diamond((x > 0) \wedge \Diamond(y > 0))$. The expressions for nominal robustness of $\Diamond(y > 0)$ and $x > 0$ are as in (5.39) and (5.35) respectively. From Lemma (3), the expression for $(x > 0) \wedge \Diamond(y > 0)$ is:

$$E\begin{pmatrix} t_i, \min(-\infty, y(n)), \\ (\min(-\infty, +\infty),\ \Diamond(y > 0)) \\ (\min(y(n), +\infty),\ x > 0) \\ (\min(+\infty, +\infty),\ (x > 0) \wedge \Diamond(y > 0)) \end{pmatrix} \tag{5.44}$$

We can drop the second line in the above expression, as it corresponds to a min with $-\infty$ as an argument. The above expression then simplifies to:

$$E\left(t_i, -\infty,\ \ (y(n),\ x > 0)\,,\ \ (+\infty,\ (x > 0) \wedge \Diamond(y > 0))\right). \tag{5.45}$$

Now, applying Lemma 5 again, we get the following expression:

$$E\begin{pmatrix} 0, -\infty, \\ (y(n),\ \Diamond(x > 0))\,, \\ (+\infty,\ \Diamond(x > 0 \wedge \Diamond(y > 0))) \end{pmatrix} \tag{5.46}$$

This simplifies to the following expression:

$$\max\left(\min(y(n), \rho_{\text{nom}}(\Diamond(x > 0), \mathbf{x}_{[0,n-1]}, 0)), \rho_{\text{nom}}(\Diamond((x > 0) \wedge \Diamond(y > 0)), \mathbf{x}_{[0,n-1]}, 0)\right).$$

Thus, the formula $\Diamond(x > 0 \wedge \Diamond(y > 0))$ can be monitored using two variables, one for the value of $\Diamond(x > 0)$ till time t_{n-1} and one for $\Diamond((x > 0) \wedge \Diamond(y > 0))$ till time t_{n-1}.

5.4 Extension to full STL

To extend the results from STL_u to STL, we need to perform certain steps. First, we recall a result from [11], which shows that any Metric Temporal Logic formula can be rewritten as a formula in which no unbounded time-horizon temporal operator appears in the scope of a bounded horizon temporal operator.

Example 6 Consider the formula $\Diamond_{[0,2]}\Box(x > 0)$. Following the rewrite rules from [11], we can rewrite this formula as: $(\Diamond_{[0,2]}\Box_{[0,2]}(x > 0)) \wedge \Box(\tilde{x} > 0)$, where $\tilde{x}$ is the signal x delayed by 2 time units.

We can now extend Theorem 1 to allow subformulae that are not just atomic predicates, but timed temporal operators. The key idea is to use a syntax tree similar to that for online monitoring of the bounded horizon STL formulae as in Sect. 4. For formulas containing unbounded horizon STL formulas, a naïve version of the algorithm would require an infinite number of elements to be stored in the worklists at each node in the tree. However, by observing that we can *summarize* computation on partial signals in variables, we can monitor STL formulas using a finite amount of memory.

For any unbounded horizon temporal formula, suppose the maximum time horizon of the bounded horizon subformulas is h. Given a partial signal from time t_0 to t_n, we can use the techniques from Theorem 1 without modification to monitor the nominal robustness till time $t_n - h$. For the last h time units, we cannot compute the exact robustness values of the bounded horizon temporal formulas. For each time-point in the interval $[t_n - h, t_n]$, we can maintain a tree with worklists similar to Algorithm 2 to monitor the bounded horizon subformula. Note that due to the minimum time-step stipulation, we need to maintain only a finite number of such trees. Thus, unlike the case for STL$_u$ where the computation of nominal robustness over the partial signal $\mathbf{x}_{[0,n]}$ can be expressed in terms of a summary value over the partial signal $\mathbf{x}_{[0,n-1]}$, and the signal value at time t_n, in case of full STL, the computation is expressed as computation over the partial signal over $[0, t_n - h]$, and a finite number of trees for the bounded horizon subformulae (which contain the signal values in $[t_n - h, t_n]$. We omit the precise details for brevity.

6 Experimental results

We implemented Algorithm 2 as part of two independant tools: Breach [4], and CPSGrader [16]. Breach is a Matlab toolbox with a strong focus on monitoring and analysis of Simulink models [20], though simple wrappers can be used to make it usable with other simulators or real time measured data. CPSGrader is a C++ library. Both tools support full STL specifications. In the following, we illustrate the use of online monitoring with the benchmark model proposed in [14], then evaluate it on two case studies. 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?

6.1 Fuel control model

The model presented in Fig. 4 is a subsystem of gasoline engine implemented in Simulink. Its purpose is to control the engine air-fuel (AF) ratio so as to meet emissions targets, an important control functionality in a gasoline engine. The model contains the air-fuel controller and a mean value model of the engine dynamics, such as the throttle and intake manifold air dynamics. Inputs of this system model are $PedalAngle$ and $EngineSpeed$. Outputs are AF, AFref and controller Mode. Typical properties to verify and monitor are overshoot and settling time.

To illustrate the use of online monitoring in Simulink, we augmented the test harness with an STL monitoring block available in Breach. The block, depicted in Fig. 5, accepts arbitrary

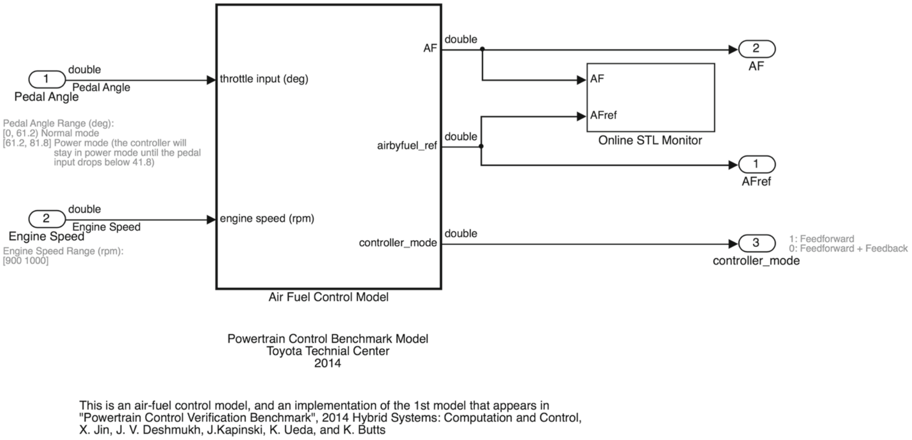

1
Pedal Angle
double
Pedal Angle
Pedal Angle Range (deg):
[0, 61.2) Normal mode
[61.2, 81.8] Power mode (the controller will stay in power mode until the pedal input drops below 41.8)
2
Engine Speed
double
Engine Speed
Engine Speed Range (rpm):
[900 1000]
throttle input (deg)
engine speed (rpm)
AF
airbyfuel_ref
controller_mode
Air Fuel Control Model
double
double
double
AF
AFref
Online STL Monitor
2
AF
1
AFref
3
controller_mode
1: Feedforward
0: Feedforward + Feedback
Powertrain Control Benchmark Model
Toyota Technial Center
2014
This is an air-fuel control model, and an implementation of the 1st model that appears in "Powertrain Control Verification Benchmark", 2014 Hybrid Systems: Computation and Control, X. Jin, J. V. Deshmukh, J.Kapinski, K. Ueda, and K. Butts

Fig. 4 Abstract Fuel Control model test harness augmented with an online monitoring block. Details of the model can be found in the given reference [14]

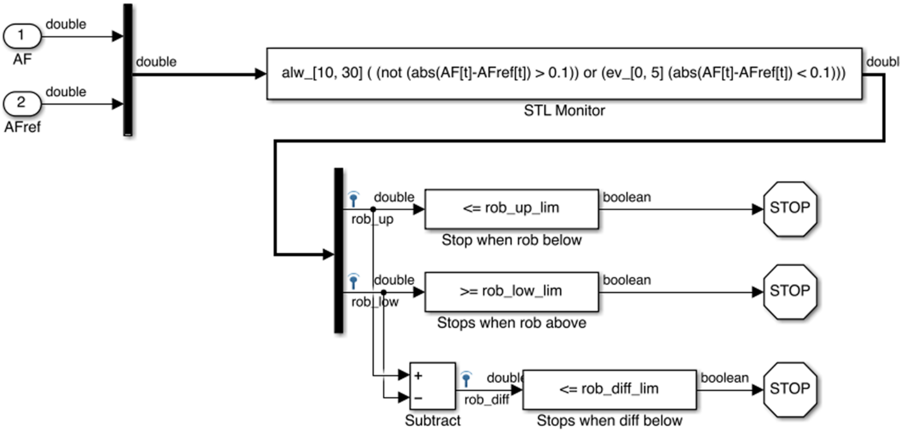


Fig. 5 Details of the online monitoring block for the Abstract Fuel Control model and formula 6.1

STL formulas on its input signals and outputs a two dimensional signals computing the robust interval. It can be configured to stop simulation whenever the robust interval becomes positive, negative or singular,i.e., when no uncertainty remain in the eventual value of robust satisfaction.

We ran the model with the following formula:

$$\varphi_{\text{AFC}} = \Box_{[10,30]}(|\text{AF} - \text{AFref}| > 0.1 \implies (\Diamond_{[0,5]}|\text{AF} - \text{AFref}| < 0.1)) \tag{6.1}$$

The resulting signals, AF, AFref and lower and upper robustness are depicted on Fig. 6. It shows the reduction of robust interval as time advances, until the full determination of its sign and value, before the expected horizon of the formula.

Next, we proceed with a more systematic evaluation of online monitoring performance on two case studies.

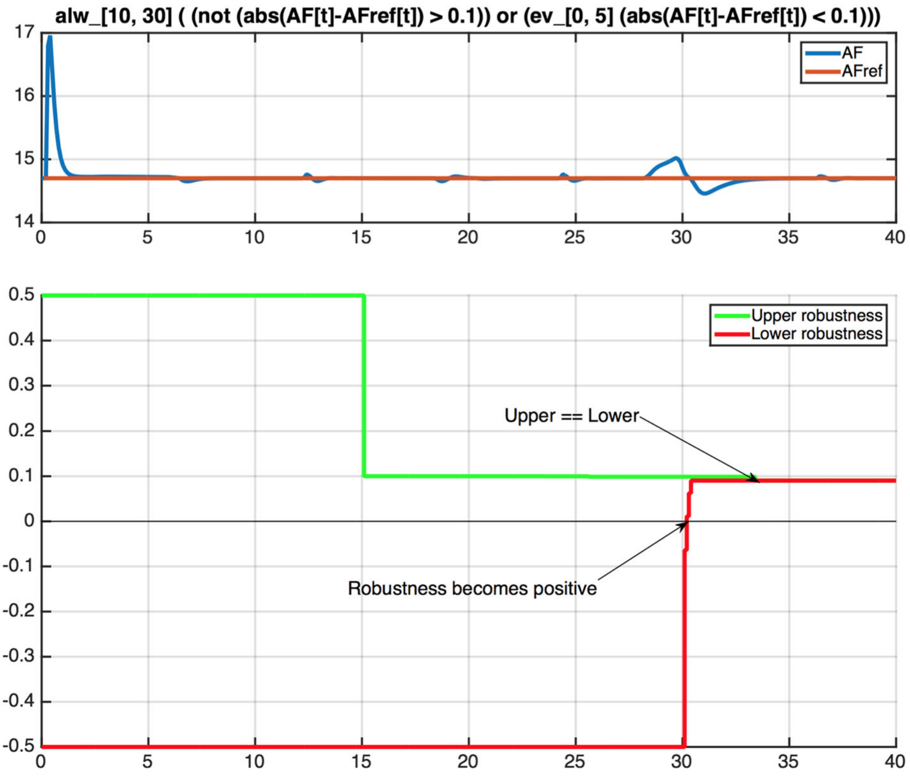


Fig. 6 Output of a simulation and online computation of *lower* and *upper* bounds of robustness for the fuel control model. Note that in this example, we set the maximum interval robustness to [−0.5, 0.5]. In this run, simulation could have been stopped early shortly after t = 30 s, when interval robustness becomes positive, or at t = 33.5 s when the interval becomes singular, i.e., robust satisfaction is deterministically known. Note that the horizon of the formula is 35 s. (Color figure online)

6.2 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 s of operation takes 5 s 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. (6.2)–(6.3)). Due to proprietary concerns, we suppress the actual values of the parameters used in the requirements.

$$\varphi_{overshoot}(\mathbf{p_1}) = \Box_{[a,b]}(\mathbf{x} < c) \tag{6.2}$$

$$\varphi_{transient}(\mathbf{p_2}) = \Box_{[a,b]}(|\mathbf{x}| > c \implies (\Diamond_{[0,d]}|\mathbf{x}| < e)) \tag{6.3}$$

Table 1 Experimental results on DEM

Requirement	Num.	Early	Simulation time (h)	
	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

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%).

6.3 CPSGrader

CPSGrader [6,16] 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 [16], 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 s of real-world operation of the system takes 1 s 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 [6,16]. These traces are the result of running actual student submissions on a battery of tests. For lack of space, we refer the reader to [16] for details about the tests and STL properties. As an illustrative example, we show the `keep_bump` property in Eq. 6.4:

$$\varphi_{\texttt{keep\_bump}} = \Diamond_{[0,60]} \Box_{[0,5]} (\texttt{bump\_right}(t) \vee \texttt{bump\_left}(t)) \qquad (6.4)$$

This formula encodes the behavior that less than 60 s from the beginning of the simulation, either the right bump sensor or the left bump sensor stays on for at least 5 s, which indicates that the robot is likely stuck against an obstacle.

Table 2 Evaluation of online monitoring for CPSGrader

STL test bench	Num.	Early	Sim. time (mins)		Overhead (s)	
	Traces	Termination	Offline	Online	Naïve	Algorithm 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

Each STL test bench has an associated STL property

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 s 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 Algorithm 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 Algorithm 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 40–80x.

7 Conclusion

STL is a convenient and versatile specification language for monitoring complex requirements or specifications for CPS. Its robust semantics provides information complementing the usual Boolean Yes/No satisfaction answer that can be instrumental at the design, verification or runtime phase. Being able to compute it online, i.e., as simulations or the actual system run, is thus a highly desirable feature, for which this paper proposes an efficient and sound solution. The key idea is the concept of interval robustness, which provides conservative bounds for the robust satisfaction of a formula by a behavior at all time instants. We showed how robust intervals can be computed efficiently and using only bounded memory. As future work, we will propose and implement new algorithms for monitoring piecewise linear signals specified and investigate other quantitative semantics, e.g., as in [22], and possible efficient hardware implementations based on FPGA, in the spirit of [13].

Acknowledgements This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

1. Annpureddy Y, Liu C, Fainekos G, Sankaranarayanan S (2011) S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: TACAS. pp 254–257
2. Bartocci E, Bortolussi L, Sanguinetti G (2014) Data-driven statistical learning of temporal logic properties. In: Formal modeling and analysis of timed systems. Springer International Publishing, pp 23–37
3. Dokhanchi A, Hoxha B, Fainekos G (2014) On-line monitoring for temporal logic robustness. In: RV. pp 231–246
4. Donzé A (2010) Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: CAV. pp 167–170
5. Donzé A, Ferrère T, Maler O (2013) Efficient robust monitoring for STL. In: CAV. pp 264–279
6. Donzé A, Juniwal G, Jensen JC, Seshia SA, Cpsgrader website. http://www.cpsgrader.org
7. Donzé A, Maler O (2010) Robust satisfaction of temporal logic over real-valued signals. In: Formal modeling and analysis of timed systems. pp 92–106
8. Eisner C, Fisman D, Havlicek J, Lustig Y, McIsaac A, Campenhout DV (2003) Reasoning with temporal logic on truncated paths. In: CAV. pp 27–39
9. Fainekos G, Sankaranarayanan S, Ueda K, Yazarel H (2012) Verification of automotive control applications using s-taliro. In: Proceedings of the American Control Conference
10. Fainekos GE, Pappas GJ (2009) Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. 410(42):4262–4291
11. Ho H-M, Ouaknine J, Worrell J (2014) Online monitoring of metric temporal logic. In: Runtime verification
12. Hoxha B, Abbas H, Fainekos G (2014) Benchmarks for temporal logic requirements for automotive systems. In: Proceedings of applied verification for continuous and hybrid systems
13. Jaksic S, Bartocci E, Grosu R, Kloibhofer R, Nguyen T, Nickovic D (2015) From signal temporal logic to FPGA monitors. In: 13. ACM/IEEE international conference on formal methods and models for codesign, MEMOCODE 2015, Austin, 21–23 Sept 2015. IEEE, pp 218–227
14. Jin X, Deshmukh JV, Kapinski J, Ueda K, Butts K (2014) Powertrain control verification benchmark. In: Proceedings of hybrid systems: computation and control. pp 253–262
15. Jin X, Donzé A, Deshmukh JV, Seshia SA (2013) Mining requirements from closed-loop control models. In: Proceedings of HSCC. pp 43–52
16. Juniwal G, Donzé A, Jensen JC, Seshia SA (2014) CPSGrader: synthesizing temporal logic testers for auto-grading an embedded systems laboratory. In: EMSOFT
17. Kong Z, Jones A, Medina Ayala A, Aydin Gol E, Belta C (2014) Temporal logic inference for classification and prediction from data. In: Proceedings of the 17th international conference on hybrid systems: computation and control. ACM, pp 273–282
18. Lemire D (2006) Streaming maximum-minimum filter using no more than three comparisons per element. arXiv preprint arXiv:cs/0610046
19. Maler O, Nickovic D (2004) Monitoring temporal properties of continuous signals. In: FORMATS/FTRTFT. pp 152–166
20. MATLAB/Simulink (2015) Version R2015a. The MathWorks Inc., Natick
21. Nickovic D, Maler O (2007) AMT: A property-based monitoring tool for analog systems. Form Model Anal Timed Syst 4763:304–319
22. Rodionova A, Bartocci E, Nickovic D, Grosu R (2016) Temporal logic as filtering. In: Abate A, Fainekos GE (eds) Proceedings of the 19th international conference on hybrid systems: computation and control, HSCC 2016, Vienna, 12–14 Apr 2016. ACM, pp 11–20