Approximate Distributed Monitoring under Partial Synchrony: Balancing Speed with Accuracy

Author(s)

Affiliation(s)

Abstract. In distributed systems with processes that do not share a global clock, *partial synchrony* is achieved by clock synchronization that guarantees bounded clock skew among all applications. Existing solutions for distributed runtime verification under partial synchrony against temporal logic specifications are exact but suffer from significant computational overhead. In this paper, we propose an *approximate* distributed monitoring algorithm for Signal Temporal Logic (STL) that mitigates this issue by abstracting away potential interleaving behaviors. This conservative abstraction enables a significant speedup of the distributed monitors, albeit with a trade-off in accuracy. We address this trade-off with a methodology that combines our approximate monitor with its exact counterpart, resulting in enhanced monitoring efficiency without sacrificing precision. We validate our approach with multiple experiments, showcasing its effectiveness and efficacy on both a real-world application and synthetic examples.

You can leave notes using the command \firstName{note}.
Page limit: 16 + 4 (appendix)

1 Introduction

Distributed systems are networks of independent agents that work together to achieve a common objective. Distributed systems are everywhere around us and come in many different forms. For example, cloud computing uses distribution of resources and services over the internet to offer to their users a scalable infrastructure with transparent on-demand access to computing power and storage. Swarms of drones represent another family of distributed systems where individual drones collaborate to accomplish tasks like surveillance, search and rescue, or package delivery. While each drone operates independently, it also communicates and coordinates with others to successfully achieve their common objectives. The individual agents in a distributed system typically do not share a global clock. To coordinate actions across multiple agents, clock synchronization is often needed. While perfect clock synchronization is impractical due to network latency and node failures, algorithms such as the Network Time Protocol (NTP) allow agents to maintain a *bounded skew* between the synchronized clocks. In that case, we say that a distributed system has *partial synchrony*.

Formal verification of distributed system is a notoriously hard problem, due to the combinatorial explosion of all possible interleavings in the behaviors collected from individual agents. *Runtime verification (RV)* provides a more pragmatic approach, in which a monitor observes a behavior of a distributed system and checks its correctness against a formal specification. The problem of distributed RV under partial synchrony assumption has been studied for Linear Temporal Logic (LTL) and Signal Temporal Logic (STL) specification languages. The proposed solutions use Satisfiability-Modulo-Theory (SMT) solving to provide sound and complete distributed monitoring procedures. Although distributed RV monitors consume only a single distributed behavior at a time, this behavior can nevertheless have an excessive number of possible interleavings. Hence, the exact distributed monitors from the literature can still suffer from significant computational overhead.

To mitigate this issue, we present an approach for *approximate* RV of STL specifications under partial synchrony. In essence, we abstract away potential interleavings in distributed behaviors in a conservative manner, resulting in an effective over-approximation of global behaviors. This abstraction simplifies the representation of distributed behaviors and the monitoring operations required to evaluate temporal specifications. There is an inevitable trade-off in approximate RV – gains in the monitoring speed-up may result in reduced accuracy. For some applications, reduced accuracy may not be acceptable. Therefore, we propose a methodology that combines our approximate monitors with their exact counterparts, with the aim to benefit from the enhanced monitoring efficiency without sacrificing precision. We implemented our approach in a prototype tool and performed thorough evaluations on both synthetic and real-world case studies. We first demonstrated that our approximate monitors achieve speed-ups of several orders of magnitudes compared to the exact SMT-based distributed RV solution. We empirically characterized the classes of specifications and behaviors for which our approximate monitoring approach achieves good precision. We finally showed that by combining exact and approximate distributed RV, there is still a significant efficiency gain on average without the sacrifice of the precision, even in cases where approximate monitors have low accuracy.

2 Preliminaries

We denote by $\mathbb{B} = \{\top, \bot\}$ the set of Booleans, $\mathbb{R}$ the set of reals, $\mathbb{R}_{\geq 0}$ the set of nonnegative reals, and $\mathbb{R}_{>0}$ the set of positive reals. An interval $I \subseteq \mathbb{R}$ of reals with the end points $a < b$ has length $|b - a|$.

Let Σ be a finite *alphabet*. We denote by Σ^* the set of finite words over Σ and by ϵ the empty word. For $u \in \Sigma^*$, we respectively write $\mathsf{prefix}(u)$ and $\mathsf{suffix}(u)$ for the sets of prefixes and suffixes of u. We also let $\mathsf{infix}(u) = \{v \in \Sigma^* \mid \exists x, y \in \Sigma^* : u = xvy\}$. For a nonempty word $u \in \Sigma^*$ and $1 \leq i \leq |u|$, we denote by $u[i]$ the ith letter of u, by $u[..i]$ the prefix of u of length i, and by

$u[i..]$ the suffix of u of length $|u| - i + 1$. Given $u \in \Sigma^*$ and $\ell \geq 1$, we denote by u^ℓ the word obtained by concatenating u by itself $\ell - 1$ times. Moreover, given $L \subseteq \Sigma^*$, we define $\mathsf{first}(L) = \{u[0] \mid u \in L\}$. For sets $L_1, L_2 \subseteq \Sigma^*$ of words, we let $L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1, v \in L_2\}$. For tuples $(u_1, \ldots, u_m)$ and $(v_1, \ldots, v_m)$ of words, we let $(u_1, \ldots, u_m) \cdot (v_1, \ldots, v_m) = (u_1 v_1, \ldots, u_m v_m)$.

We define the function $\mathsf{destutter} : \Sigma^* \to \Sigma^*$ inductively as follows. For all $\sigma \in \Sigma \cup \{\epsilon\}$, let $\mathsf{destutter}(\sigma) = \sigma$. For all $u \in \Sigma^*$ such that $u = \sigma_1 \sigma_2 v$ for some $\sigma_1, \sigma_2 \in \Sigma$ and $v \in \Sigma^*$, let (i) $\mathsf{destutter}(u) = \mathsf{destutter}(\sigma_2 v)$ if $\sigma_1 = \sigma_2$, and (ii) $\mathsf{destutter}(u) = \sigma_1 \cdot \mathsf{destutter}(\sigma_2 v)$ otherwise. By extension, for a set $L \subseteq \Sigma^*$ of finite words, we write $\mathsf{destutter}(L) = \{\mathsf{destutter}(u) \mid u \in L\}$. Given a tuple $(u_1, \ldots, u_m) = (\sigma_{1,1}\sigma_{1,2}v_1, \ldots, \sigma_{m,1}\sigma_{m,2}v_m)$ of finite words of the same length, we define $\mathsf{destutter}(u_1, \ldots, u_m)$ as expected: (i) $\mathsf{destutter}(u_1, \ldots, u_m) = \mathsf{destutter}(\sigma_{1,2}v_1, \ldots, \sigma_{m,2}v_m)$ if $\sigma_{i,1} = \sigma_{i,2}$ for all $1 \leq i \leq m$, and (ii) $\mathsf{destutter}(u_1, \ldots, u_m) = (\sigma_{1,1}, \ldots, \sigma_{m,1}) \cdot \mathsf{destutter}(\sigma_{1,2}v_1, \ldots, \sigma_{m,2}v_m)$ otherwise.

Moreover, given an integer $k \geq 0$, we define $\mathsf{stutter}_k : \Sigma^* \to \Sigma^*$ such that $\mathsf{stutter}_k(u) = \{v \in \Sigma^* \mid |v| = k \wedge \mathsf{destutter}(v) = \mathsf{destutter}(u)\}$ if $k \geq |\mathsf{destutter}(u)|$, and $\mathsf{stutter}_k(u) = \emptyset$ otherwise.

Signal Temporal Logic (STL) [1]. Let $A, B \subset \mathbb{R}$. A function $f : A \to B$ is *right-continuous* iff $\lim_{a \to c^+} f(a) = f(c)$ for all $c \in A$, and *non-Zeno* iff for every bounded interval $I \subseteq A$ there are finitely many $a \in I$ such that f is not continuous at a. A *signal* is a right-continuous, non-Zeno, piecewise-constant function $x : [0, d) \to \mathbb{R}$ where $d \in \mathbb{R}_{>0}$ is the duration of x and $[0, d)$ is its temporal domain. Let $x : [0, d) \to \mathbb{R}$ be a signal. An *event* of x is a pair $(t, x(t))$ where $t \in [0, d)$. An *edge* of x is an event $(t, x(t))$ such that $\lim_{s \to t^-} x(s) \neq \lim_{s \to t^+} x(s)$. In particular, an edge is *rising* if $\lim_{s \to t^-} x(s) < \lim_{s \to t^+} x(s)$, and it is *falling* otherwise. A signal $x : [0, d) \to \mathbb{R}$ can be represented finitely by its initial value and edges: if x has m edges, then $x = (t_0, v_0)(t_1, v_1) \ldots (t_m, v_m)$ such that $t_0 = 0$, $t_{i-1} < t_i$, and (t_i, v_i) is an edge of x for all $1 \leq i \leq m$.

Let AP be a set of *atomic propositions*. The syntax is given by the following grammar where $p \in \mathsf{AP}$ and $I \subseteq \mathbb{R}_{\geq 0}$ is an interval.

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U}_I \varphi$$

A *trace* $w = (x_1, \ldots, x_n)$ is a finite vector of signals. We express atomic propositions as functions of trace values at a time point t, i.e., a proposition $p \in \mathsf{AP}$ over a trace $w = (x_1, \ldots, x_n)$ is defined as $f_p(x_1(t), \ldots, x_n(t)) > 0$ where $f_p : \mathbb{R}^n \to \mathbb{R}$ is a function. Given intervals $I, J \subseteq \mathbb{R}_{\geq 0}$, we define $I \oplus J = \{i + j \mid i \in I \wedge j \in J\}$, and we simply write t for the singleton set $\{t\}$.

Below, we recall the finite-trace qualitative semantics of STL defined over $\mathbb{B}$ [1]. Let $d \in \mathbb{R}_{>0}$ and $w = (x_1, \ldots, x_n)$ with $x_i : [0, d) \to \mathbb{R}$ for all $1 \leq i \leq n$. Let φ_1, φ_2 be STL formulas and let $t \in [0, d)$.

$$
\begin{aligned}
(w,t) \models p &\iff f_p(x_1(t),\ldots,x_n(t)) > 0 \\
(w,t) \models \neg\varphi_1 &\iff \overline{(w,t) \models \varphi_1} \\
(w,t) \models \varphi_1 \wedge \varphi_2 &\iff (w,t) \models \varphi_1 \wedge (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \mathcal{U}_I \varphi_2 &\iff \exists t' \in (t \oplus I) \cap [0,d) : \\
&\qquad (w,t') \models \varphi_2 \wedge \forall t'' \in (t,t') : (w,t'') \models \varphi_1
\end{aligned}
$$

We simply write $w \models \varphi$ for $(w,0) \models \varphi$. We additionally use the following standard abbreviations: $\mathtt{false} = p \wedge \neg p$, $\mathtt{true} = \neg\mathtt{false}$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\Diamond_I \varphi = \mathtt{true}\mathcal{U}_I\varphi$, and $\Box_I \varphi = \neg \Diamond_I \neg\varphi$. Moreover, the untimed temporal operators are defined through their timed counterparts on the interval $[0,\infty)$, e.g., $\varphi_1 \mathcal{U} \varphi_2 = \varphi_1 \mathcal{U}_{[0,\infty)} \varphi_2$.

Distributed Semantics of STL [2]. We consider an asynchronous and loosely-coupled message-passing system of $n \geq 2$ reliable agents producing a set of signals $x_1,\ldots,x_n$, where for some $d \in \mathbb{R}_{>0}$ we have $x_i : [0,d) \to \mathbb{R}$ for all $1 \leq i \leq n$. The agents do not share memory or a global clock. Only to formalize statements, we speak of a *hypothetical* global clock and denote its value by T. For local time values, we use the lowercase letters t and s.

For a signal x_i, we denote by V_i the set of its events, by $E_i^{\uparrow}$ the set of its rising edges, and by $E_i^{\downarrow}$ that of falling edges. Moreover, we let $E_i = E_i^{\uparrow} \cup E_i^{\downarrow}$. We represent the local clock of the ith agent as an increasing and divergent function $c_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that maps a global time T to a local time $c_i(T)$.

We assume that the system is *partially synchronous*: the agents use a clock synchronization algorithm that guarantees a bounded clock skew with respect to the global clock, i.e., $|c_i(T) - c_j(T)| < \varepsilon$ for all $1 \leq i,j \leq N$ and $T \in \mathbb{R}_{\geq 0}$, where $\varepsilon \in \mathbb{R}_{>0}$ is the maximum clock skew.

Definition 1. *A* distributed signal *is a pair $(S, \rightsquigarrow)$, where $S = (x_1,\ldots,x_n)$ is a vector of signals and $\rightsquigarrow$ is the happened-before relation between events in signals extended with the partial synchrony assumption as follows.*

- *For every agent, the events of its signals are totally ordered, i.e., for all $1 \leq i \leq n$ and all $(t, x_i(t)), (t', x_i(t')) \in V_i$, if $t < t'$ then $(t, x_i(t)) \rightsquigarrow (t', x_i(t'))$.*
- *Every pair of events whose timestamps are at least ε apart is totally ordered, i.e., for all $1 \leq i,j \leq n$ and all $(t, x_i(t)) \in V_i$ and $(t', x_j(t')) \in V_j$, if $t+\varepsilon \leq t'$ then $(t, x_i(t)) \rightsquigarrow (t', x_j(t'))$.*

Example 2. TODO: distributed signal, happened-before relation

Definition 3. *Let $(S, \rightsquigarrow)$ be a distributed signal of n signals, and $V = \bigcup_{i=1}^{n} V_i$ be the set of its events. A set $C \subseteq V$ is a* consistent cut *iff for every event in C, all events that happened before it also belong to C, i.e., for all $e, e' \in V$, if $e \in C$ and $e' \rightsquigarrow e$, then $e' \in C$.*

We denote by $\mathbb{C}(T)$ the (infinite) set of consistent cuts at global time T. Given a consistent cut C, its *frontier* $\mathsf{front}(C) \subseteq C$ is the set consisting of the last events in C of each signal, i.e., $\mathsf{front}(C) = \bigcup_{i=1}^{n} \{(t, x_i(t)) \in V_i \cap C \mid \forall t' > t : (t', x_i(t')) \notin V_i \cap C\}$.

Definition 4. *A* consistent cut flow *is a function* $\mathsf{ccf} : \mathbb{R}_{\geq 0} \to 2^V$ *that maps a global clock value* T *to the frontier of a consistent cut at time* T*, i.e.,* $\mathsf{ccf}(T) \in \{\mathsf{front}(C) \mid C \in \mathbb{C}(T)\}$.

For all $T, T' \in \mathbb{R}_{\geq 0}$ and $1 \leq i \leq n$, if $T < T'$, then for every pair of events $(c_i(T), x_i(c_i(T))) \in \mathsf{ccf}(T)$ and $(c_i(T'), x_i(c_i(T'))) \in \mathsf{ccf}(T')$ we have $(c_i(T), x_i(c_i(T))) \rightsquigarrow (c_i(T'), x_i(c_i(T')))$. We denote by $\mathsf{CCF}(S, \rightsquigarrow)$ the set of all consistent cut flows of the distributed signal $(S, \rightsquigarrow)$.

Example 5. TODO: consistent cut, frontier, consistent cut flow

Observe that a consistent cut flow of a distributed signal induces a vector of synchronous signals which can be evaluated using the standard semantics described in Section 2. Let $(S, \rightsquigarrow)$ be a distributed signal of n signals $x_1, \ldots, x_n$. A consistent cut flow $\mathsf{ccf} \in \mathsf{CCF}(S, \rightsquigarrow)$ yields a trace $w_{\mathsf{ccf}} = (x'_1, \ldots x'_n)$ on the temporal domain $[0, d)$ such that $(c_i(T), x_i(c_i(T))) \in \mathsf{ccf}(T)$ implies $x'_i(T) = x_i(c_i(T))$ for all $1 \leq i \leq n$ and $T \in [0, d)$. The set of traces of $(S, \rightsquigarrow)$ is given by $\mathsf{Tr}(S, \rightsquigarrow) = \{w_{\mathsf{ccf}} \mid \mathsf{ccf} \in \mathsf{CCF}(S, \rightsquigarrow)\}$.

We define the satisfaction of an STL formula φ by a distributed signal $(S, \rightsquigarrow)$ over a three-valued domain $\{\top, \bot, ?\}$ If the set of synchronous traces $\mathsf{Tr}(S, \rightsquigarrow)$ defined by a distributed signal $(S, \rightsquigarrow)$ is contained in the set of traces allowed by the formula φ, then $(S, \rightsquigarrow)$ satisfies φ. Similarly, if $\mathsf{Tr}(S, \rightsquigarrow)$ has an empty intersection with the set of traces φ defines, then $(S, \rightsquigarrow)$ violates φ. Otherwise, the evaluation is inconclusive since some traces satisfy the property and some violate it. Notice that we quantify universally over traces for both satisfaction and violation.

$$[(S, \rightsquigarrow) \models \varphi] = \begin{cases} \top & \text{if } \forall w \in \mathsf{Tr}(S, \rightsquigarrow) : w \models \varphi \\ \bot & \text{if } \forall w \in \mathsf{Tr}(S, \rightsquigarrow) : w \models \neg\varphi \\ ? & \text{otherwise} \end{cases}$$

3 Overapproximation of the STL Distributed Semantics

To address the computational overhead in exact distributed monitoring, we define STL$^+$, a variant of STL whose syntax is the same as STL but semantics provide a sound approximation of the STL distributed semantics. STL$^+$ is better adapted to distributed monitoring as it overapproximates the set of traces. In particular, given a distributed signal $(S, \rightsquigarrow)$, STL$^+$ considers an approximation $\mathsf{Tr}^+(S, \rightsquigarrow)$ of the set $\mathsf{Tr}(S, \rightsquigarrow)$ of synchronous traces where $\mathsf{Tr}(S, \rightsquigarrow) \subseteq \mathsf{Tr}^+(S, \rightsquigarrow)$. A signal $(S, \rightsquigarrow)$ satisfies (resp. violates) an STL$^+$ formula φ iff all the traces in $\mathsf{Tr}^+(S, \rightsquigarrow)$ belong to the language of φ (resp. $\neg\varphi$).

$$[(S, \rightsquigarrow) \models \varphi]_+ = \begin{cases} \top & \text{if } \forall w \in \mathsf{Tr}^+(S, \rightsquigarrow) : w \models \varphi \\ \bot & \text{if } \forall w \in \mathsf{Tr}^+(S, \rightsquigarrow) : w \models \neg\varphi \\ ? & \text{otherwise} \end{cases}$$

In Sections 4 and 5, we respectively define Tr^+ and present an algorithm to compute the semantics of STL$^+$. We finally prove the correctness of our approach.

Theorem 6. *For every STL formula φ and every distributed signal $(S, \rightsquigarrow)$, if $[(S, \rightsquigarrow) \models \varphi]_+ = \top$ (resp. $\bot$) then $[(S, \rightsquigarrow) \models \varphi] = \top$ (resp. $\bot$).*

Notice that both the distributed semantics of STL and the semantics of STL$^+$ quantify universally over the set of traces for the verdicts $\top$ and $\bot$. Therefore, the Theorem 6 holds for the verdicts $\top$ and $\bot$, but not for ?.

4 Overapproximation of Synchronous Traces

In this section, given a distributed signal $(S, \rightsquigarrow)$, we describe an overapproximation $\mathsf{Tr}^+(S, \rightsquigarrow)$ of its set $\mathsf{Tr}(S, \rightsquigarrow)$ of synchronous traces. First, we present the notion of *canonical segmentation*, a systematic way of partitioning the temporal domain of a given distributed signal to keep track of the partial asynchrony. Second, we introduce the notion of *value expressions*, sets of finite words representing how a signal behaves in a time interval. Finally, we define Tr^+ based on these notions, and show that it soundly approximates Tr.

Remark 7. We assume boolean signals in this section for convenience. The definitions and results presented here extend to real-valued signals because finite-length piecewise-constant signals will only use a finite number of values.

Canonical Segmentation Consider a boolean signal x with a rising edge at time $t > \varepsilon$. Due to clock skew, this edge occurs in the range $(t - \varepsilon, t + \varepsilon)$ from the monitor's point of view. This range is called an *uncertainty region* because in $(t - \varepsilon, t + \varepsilon)$ the monitor cannot tell the value of x precisely, but only that it changes from 0 to 1. Formally, given an edge $(t, x(t))$, we define $\theta_{\text{lo}}(x, t) = \max(0, t - \varepsilon)$ and $\theta_{\text{hi}}(x, t) = \min(d, t + \varepsilon)$ as the end points of the edge's uncertainty region.

Given a temporal domain $I = [0, d) \subset \mathbb{R}_{\geq 0}$, a *segmentation* of I is a partition of I into finitely many intervals $I_1, \dots, I_k$, called *segments*, of the form $I_j = [t_j, t_{j+1})$ such that $t_j < t_{j+1}$ for all $1 \leq j \leq k$. By extension, a segmentation of a collection of signals with the same temporal domain I is a segmentation of I.

Let $(S, \rightsquigarrow)$ be a distributed signal of n signals. The *canonical segmentation* G_S of $(S, \rightsquigarrow)$ is the segmentation of S where the end points of the segments coincide with the end points of its temporal domain and uncertainty regions. Formally, we define G_S as follows. For each signal x_i, let F_i be the set of end points of its uncertainty regions. Let $F = \{0, d\} \cup \bigcup_{i=1}^{n} F_i$ and let $(s_j)_{1 \leq j \leq |F|}$

be a nondecreasing sequence of clock values corresponding to the elements of F. Then, the canonical segmentation of $(S, \rightsquigarrow)$ is $G_S = \{I_1, \ldots, I_{|F|-1}\}$ where $I_j = [s_j, s_{j+1})$ for all $1 \leq j < |F|$.

Example 8. Let $(S, \rightsquigarrow)$ be a distributed boolean signal with $S = (x_1, x_2)$ and $\varepsilon = 2$ over the temporal domain $[0, 8)$ as given in Figure 1. Both signals are initially 0. The signal x_1 has a rising edge at time 2 and a falling edge at time 5, while x_2 has a rising edge at time 3 and a falling edge at time 6. The uncertainty regions of x_1 are $(0, 4)$ and $(3, 7)$, while those of x_2 are $(1, 5)$ and $(4, 8)$. Then, we have $F = \{0, 8\} \cup \{0, 1, 3, 4, 5, 7, 8\}$, and thus the canonical segmentation is $G_S = \{[0, 1), [1, 3), [3, 4), [4, 5), [5, 7), [7, 8)\}$.

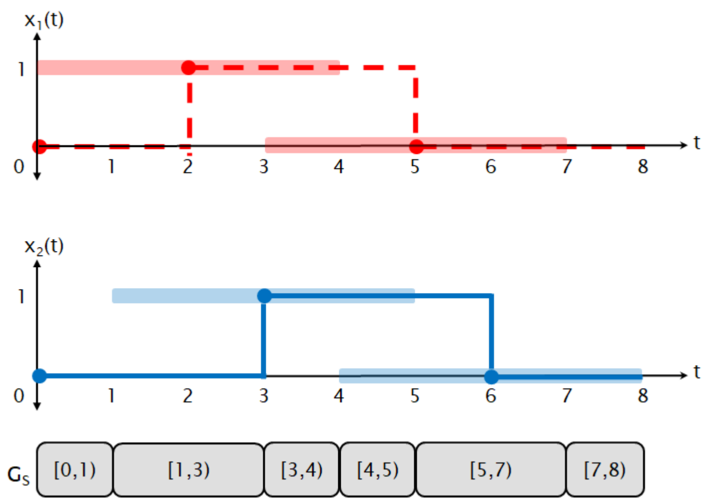

Fig. 1. The signals x_1 (top, red, dashed) and x_2 (bottom, blue, solid) from Example 8. The edges are marked with solid balls and their uncertainty regions are given as semi-transparent boxes around the edges. The resulting canonical segmentation G_S is shown below the graphical representation of the signals.

Value Expressions Consider a boolean signal x with a rising edge with an uncertainty region of (t_1, t_2). As discussed above, the monitor only knows that the value of x changes from 0 to 1 in this interval. We represent this knowledge as a finite word $v = 0 \cdot 1$. This representation is called a *value expression* and it encodes the uncertain behavior of an observed signal relative to the monitor. Formally, a value expression is an element of Σ^* where Σ is the finite alphabet of values the signal takes. Given a signal x and an edge $(t, x(t))$, the value expression corresponding to the uncertainty region $(\theta_{\text{lo}}(x, t), \theta_{\text{hi}}(x, t))$ is given by $v_{x,t} = v_- \cdot v_+$ where $v_- = \lim_{s \to t^-} x(s)$ and $v_+ = \lim_{s \to t^+} x(s)$. Let us remark that this definition is general because finite-length piecewise-constant real-valued signals will only have a finite number of values, making Σ finite.

Notice that (i) uncertainty regions may overlap, and (ii) the canonical segmentation may split an uncertainty region into multiple segments. Consider a signal x with a rising edge in $(1, 5)$ and a falling edge in $(4, 8)$. The corresponding value expressions are respectively $v_1 = 0 \cdot 1$ and $v_2 = 1 \cdot 0$. Notice that the behavior of x in the interval $[1, 4)$ can be expressed as $\mathsf{prefix}(v_1)$, encoding whether the rising edge has happened yet or not. Similarly, the behavior in $[4, 5)$ is given by $\mathsf{suffix}(v_1) \cdot \mathsf{prefix}(v_2)$, which captures whether the edges occur in this interval (thanks to prefixing and suffixing) and the fact that the rising edge happens before the falling edge (thanks to concatenation).

Formally, given a distributed signal $(S, \rightsquigarrow)$, we define a function $\gamma : S \times G_S \to 2^{\Sigma^*}$ that maps each signal and segment of the canonical segmentation to a set of value expressions, capturing the signal's potential behaviors in the given segment. Let x be a signal in S, and let $R_1, \ldots, R_m$ be its uncertainty regions where $R_i = (t_i, t'_i)$ and the corresponding value expression is v_i for all $1 \le i \le m$. Now, let $I \in G_S$ be a segment with $I = [s, s')$ and for each $1 \le i \le m$ define the set V_i of value expressions capturing how I relates with R_i as follows:

$$V_i = \begin{cases} \{v_i\} & \text{if } t_i = s \wedge s' = t'_i \\ \mathsf{prefix}(v_i) & \text{if } t_i = s \wedge s' < t'_i \\ \mathsf{suffix}(v_i) & \text{if } t_i > s \wedge s' = t'_i \\ \mathsf{infix}(v_i) & \text{if } t_i > s \wedge s' < t'_i \\ \{\epsilon\} & \text{otherwise} \end{cases} \tag{1}$$

The last case happens only when $I \cap R_i$ is empty. We finally define γ as follows:

$$\gamma(x, I) = \mathsf{destutter}(V_1 \cdot V_2 \cdot \ldots \cdot V_m) \setminus \{\epsilon\}$$

Observe that $\gamma(x, I)$ contains all the potential behaviors of x in segment I by construction. However, it is potentially overapproximate. This is mainly because the sets $V_1, \ldots, V_m$ contain redundancy by definition and the concatenation does not guarantee that an edge is considered exactly once.

Example 9. Recall the distributed signal $(S, \rightsquigarrow)$ in Example 8 and Figure 1. In Figure 2a, we show the value expressions corresponding to its uncertainty regions. For example, the falling edge of x_1 has an uncertainty region of $(3, 7)$, represented by the value expression $1 \cdot 0$. In Figure 2b, we give the function γ for $(S, \rightsquigarrow)$. For example, $\gamma(x_1, [3, 4))$ is obtained from $\mathsf{suffix}(0 \cdot 1) \cdot \mathsf{prefix}(1 \cdot 0)$ and $\gamma(x_2, [0, 1)) = \{0\}$.

Overapproximation of Tr Consider a distributed signal $(S, \rightsquigarrow)$ of n signals, and let G_S be its canonical segmentation. We describe how the function γ defines a set $\mathsf{Tr}^+(S, \rightsquigarrow)$ of synchronous traces that overapproximates the set $\mathsf{Tr}(S, \rightsquigarrow)$.

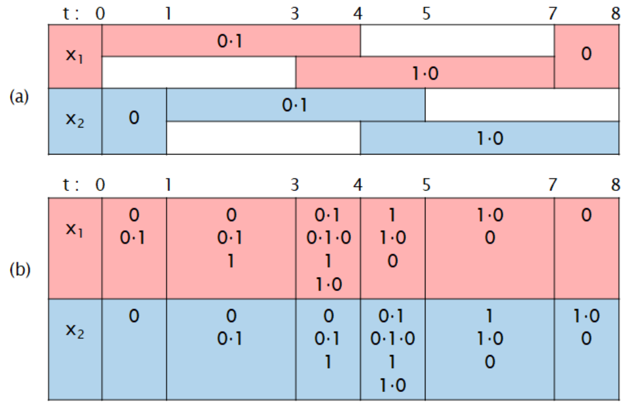

Fig. 2. (a) The uncertainty regions of the distributed signal in Example 8 and the corresponding value expressions. (b) The tabular representation of the function γ for the given distributed signal.

Let $x \in S$ and x' be two signals with the same temporal domain, and let $I = [s, s')$ be a segment in G_S. Let $(t_1, x'(t_1)), \ldots, (t_\ell, x'(t_\ell))$ be the edges of x' in segment I with $t_i < t_{i+1}$ for all $1 \leq i < \ell$. The signal x' is *I-consistent with* x iff the value expression $x'(s) \cdot x'(t_1) \cdot \ldots \cdot x'(t_\ell)$ belongs to $\gamma(x, I)$. Moreover, x' is *consistent with* x iff it is I-consistent with x for all $I \in G_S$.

Now, let $S = (x_1, \ldots, x_n)$ and define $\mathsf{Tr}^+(S, \rightsquigarrow)$ as follows:

$$\mathsf{Tr}^+(S, \rightsquigarrow) = \{(x'_1, \ldots, x'_n) \mid x'_i \text{ is consistent with } x_i \text{ for all } 1 \leq i \leq n\}$$

Example 10. Recall the distributed signal $(S, \rightsquigarrow)$ in Example 8 whose γ function is given in Figure 2b. Consider the synchronous trace $w \in \mathsf{Tr}(S, \rightsquigarrow)$ where the rising edges of both signals occur at time 3 and the falling edges at time 5. One can verify that $w \in \mathsf{Tr}^+(S, \rightsquigarrow)$ since for each $i \in \{1, 2\}$ the value expression 1 is contained in $\gamma(x_i, [3, 4))$ and $\gamma(x_i, [4, 5))$ while 0 is contained in the remaining sets γ maps x_i to.

Now, consider a synchronous trace (x'_1, x'_2) where both signals are initially 0, have rising edges at time 2 and 3.5, and falling edges at time 3 and 5. Evidently, this trace does not belong to $\mathsf{Tr}(S, \rightsquigarrow)$ since x'_1 and x'_2 have more edges than x_1 and x_2. Nonetheless, it belongs to $\mathsf{Tr}^+(S, \rightsquigarrow)$ since x'_1 and x'_2 are respectively consistent with x_1 and x_2. To witness, notice that for each $i \in \{1, 2\}$ the value expression $0 \cdot 1$ is contained in $\gamma(x_i, [1, 3))$ and $\gamma(x_i, [3, 4))$, the expression 1 is contained in $\gamma(x_i, [4, 5))$, and 0 is contained in the remaining sets γ maps x_i to.

Finally, we prove Tr^+ overapproximates Tr.

Lemma 11. *For every distributed signal $(S, \rightsquigarrow)$, we have $\mathsf{Tr}(S, \rightsquigarrow) \subseteq \mathsf{Tr}^+(S, \rightsquigarrow)$.*

5 Monitoring Algorithm

In this section, given a distributed signal $(S, \rightsquigarrow)$, we describe an algorithm to compute $[(S, \rightsquigarrow) \models \varphi]_+$. The algorithm makes use of the function γ defined in Section 4 without explicitly computing $\mathsf{Tr}^+(S, \rightsquigarrow)$. To achieve this, we first describe the notion of *asynchronous product* of value expressions to capture potential interleavings within segments. We continue with the evaluation of *untimed operators* and then *timed operators*. Finally, we conclude with putting all these together to compute the *semantics* of STL$^+$ and discuss an efficient implementation of the monitoring algorithm using *bit vectors* to represent and manipulate sets of boolean value expressions.

Remark 12. For the sake of convenience, we focus on boolean signals for the rest of the section. Note that asynchronous products and the algorithm to compute $[(S, \rightsquigarrow) \models \varphi]_+$ can be extended to value expressions over arbitrary finite alphabets, e.g., encoding real-valued signals. This allows us to express more complex properties where atomic propositions can be functions of real-valued signals.

Asynchronous Products Consider the value expressions $u_1 = 0 \cdot 1$ and $u_2 = 1 \cdot 0$ encoding the behaviors of two signals within a segment. Due to partial asynchrony, the behaviors within segments can be seen as completely asynchronous. To capture the potential interleavings of these behaviors, we consider how the values in u_1 and u_2 can align. In particular, there are three potential alignments: (i) the rising edge of u_1 happens before the falling edge of u_2, (ii) the falling edge of u_2 happens before the rising edge of u_1, and (iii) the two edges happen simultaneously. We respectively represent these with the tuples $(011, 110)$, $(001, 100)$, and $(01, 10)$ where the first component encodes u_1 and the second u_2. Formally, given two value expressions u_1 and u_2, we define their *asynchronous product* as follows:

$$u_1 \otimes u_2 = \{\mathsf{destutter}(v_1, v_2) \mid v_i \in \mathsf{stutter}_k(u_i), k = |u_1| + |u_2| - 1, i \in \{1, 2\}\}$$

Moreover, given two sets L_1 and L_2 of value expressions, we define the following:

$$L_1 \otimes L_2 = \{u_1 \otimes u_2 \mid u_1 \in L_1, u_2 \in L_2\}$$

Asynchronous products of value expressions allow us to lift value expressions to satisfaction signals of formulas.

Example 13. Recall the distributed signal $(S, \rightsquigarrow)$ in Example 8 and its γ function given in Figure 2b. Suppose we want to compute the value expressions encoding the satisfaction of $x_1 \wedge x_2$ in the segment $[1, 3)$. We can achieve this by first computing the asynchronous product $\gamma(x_1, [3, 4)) \otimes \gamma(x_2, [3, 4))$, and then computing the bitwise conjunction of each pair in the set. For example, considering the expression $0 \cdot 1 \cdot 0$ for x_1 and $0 \cdot 1$ for x_2, the product contains the pair $(010, 011)$. Taking the bitwise conjunction of this pair gives us the expression $0 \cdot 1 \cdot 0$ as a potential behavior for the satisfaction of $x_1 \wedge x_2$ in this segment.

Untimed Operations As hinted in Example 13, to compute the semantics, we apply bitwise operations on value expressions and their asynchronous products to transform them into encodings of satisfaction signals of formulas. Consider the distributed signal $(S, \rightsquigarrow)$ in Example 8 and suppose we want to compute $[(S, \rightsquigarrow) \models \Diamond(x_1 \wedge x_2)]_+$. To achieve this, we first compute for each segment in G_S the set of value expressions for the satisfaction of $x_1 \wedge x_2$, and then from these compute that of $\Diamond(x_1 \wedge x_2)$. This compositional approach allows us to evaluate arbitrary STL$^+$ formulas.

First, we define bitwise operations on boolean value expressions encoding atomic propositions. Then, we use these to evaluate (untimed) STL formulas over sets of value expressions.

Let u and v be boolean value expressions of length ℓ. We denote by $u \,\&\, v$ the bitwise-and operation, by $u \mid v$ the bitwise-or, and by $\sim u$ the bitwise-negation. In addition, we define the *bitwise strong until* operator as follows:

$$u \mathsf{U}^0 v = \left(\max_{i \leq j \leq \ell} \left(\min \left(v[j], \min_{i \leq k \leq j} u[k] \right) \right) \right)_{1 \leq i \leq \ell}$$

As usual, we derive *bitwise eventually* as $\mathsf{E}u = 1^\ell \mathsf{U}^0 u$, *bitwise always* as $\mathsf{A}u = \sim(\mathsf{E}\sim u)$, and *bitwise weak until* as $u\mathsf{U}^1 v = (u\mathsf{U}^0 v) \mid (\mathsf{A}u)$. The distinction between U^0 and U^1 will be useful later when we evaluate a formula segment by segment. We remark that the definitions of these operators coincide with the robustness semantics of (discrete time) STL. Finally, note that the output of these operations is a value expression of length ℓ. For example, if $u = 010$, we have $\mathsf{E}u = 110$ and $\mathsf{A}u = 000$.

Let $(S, \rightsquigarrow)$ be a distributed signal. Consider an atomic proposition $p \in \mathsf{AP}$ encoded as $x_p \in S$ and let φ_1, φ_2 be two STL formulas. We define the evaluation of untimed formulas with respect to $(S, \rightsquigarrow)$ and a segment $I \in G_S$ inductively:

$$\begin{aligned}
[\![(S, \rightsquigarrow), I \models p]\!] &= \gamma(x_p, I) \\
[\![(S, \rightsquigarrow), I \models \neg\varphi_1]\!] &= \{\sim u \mid u \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!]\} \\
[\![(S, \rightsquigarrow), I \models \varphi_1 \wedge \varphi_2]\!] &= \{u_1 \,\&\, u_2 \mid (u_1, u_2) \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!] \otimes [\![(S, \rightsquigarrow), I \models \varphi_2]\!]\} \\
[\![(S, \rightsquigarrow), I \models \varphi_1 \mathcal{U} \varphi_2]\!] &= \{u_1 \mathsf{U}^a u_2 \mid (u_1, u_2) \in [\![(S, \rightsquigarrow), I \models \varphi_1]\!] \otimes [\![(S, \rightsquigarrow), I \models \varphi_2]\!], \\
&\qquad a \in \mathsf{first}([\![(S, \rightsquigarrow), I' \models \varphi_1 \mathcal{U} \varphi_2]\!])\}
\end{aligned}$$

where I' is the segment that follows I in G_S, if it exists. For completeness, for every formula φ we define $[\![(S, \rightsquigarrow), I' \models \varphi]\!] = \{0\}$ when $I' \notin G_S$. When I is the first segment in G_S, we simply write $[\![(S, \rightsquigarrow) \models \varphi]\!]$. Similarly as above, we can use the standard derived operators to compute the corresponding sets of value expressions. Intuitively, for a given formula and a segment, the evaluation above produces a set of value expressions enconding the formula's satisfaction within the segment.

Example 14. Recall the distributed signal $(S, \rightsquigarrow)$ in Example 8 and its γ function given in Figure 2b. Suppose we want to compute $[\![(S, \rightsquigarrow), [5, 7) \models \Diamond(x_1 \wedge x_2)]\!]$.

First, we compute $[\![(S, \rightsquigarrow), [5,7) \models x_1 \wedge x_2]\!]$ by computing the bitwise conjunction over the asynchronous product $\gamma(x_1, [5,7)) \otimes \gamma(x_2, [5,7))$ and destuttering. For example, since $010 \in \gamma(x_1, [5,7))$ and $01 \in \gamma(x_2, [5,7))$, the pair $(0010, 0111)$ is in the product, whose conjunction gives us 010 after destuttering. Repeating this for the rest, we obtain $[\![(S, \rightsquigarrow), [5,7) \models x_1 \wedge x_2]\!] = \{0, 01, 010, 1, 10\}$. Finally, we compute $[\![(S, \rightsquigarrow), [5,7) \models \Diamond(x_1 \wedge x_2)]\!]$ by applying each expression in $[\![(S, \rightsquigarrow), [5,7) \models x_1 \wedge x_2]\!]$ the bitwise eventually operator and destuttering. The resulting set $\{0, 1, 10\}$ encondes the satisfaction signal of $\Diamond(x_1 \wedge x_2)$ in $[5,7)$. Note that we do not need to consider the evaluation of the next segment for the eventually operator since $[\![(S, \rightsquigarrow), [7,8) \models x_1 \wedge x_2]\!] = \{0\}$.

Timed Operations Handling timed operations requires a closer inspection as value expressions are untimed by definition. We address this issue by considering how a given evaluation interval relates with a given segmentation. For example, take a segmentation $G_S = \{[0,4), [4,6), [6,10)\}$ and an evaluation interval $J = [0,5)$. Suppose we are interested in how a signal $x \in S$ behaves with respect to J over the first segment $I = [0,4)$. First, to see how J relates with G_S with respect to $I = [0,4)$, we "slide" the interval J over $I \oplus J = [0,9)$ and consider the different ways it intersects the segments in G_S. Initially, J covers the entire segment $[0,4)$ and the beginning of $[4,6)$, for which the potential behaviors of x are captured by the set $\gamma(x, [0,4)) \cdot \mathsf{prefix}(\gamma(x, [4,6)))$. Now, if we slide the window and take $J' = [3,7)$, the window covers the ending of $[0,4)$, the entire $[4,6)$, and the beginning of $[6,10)$, for which the potential behaviors are captured by the set $\mathsf{suffix}(\gamma(x, [0,4))) \cdot \gamma(x, [4,6)) \cdot \mathsf{prefix}(\gamma(x, [6,9))$. We call these sets the *profiles* of J and J' with respect to $(S, \rightsquigarrow)$, x, and I.

Let $(S, \rightsquigarrow)$ be a distributed signal, $I \in G_S$ be a segment, and φ be an STL formula. Let us introduce the notation we use in the definition below. First, we abbreviate the set $[\![(S, \rightsquigarrow), I \models \varphi]\!]$ of value expressions as $\tau_{\varphi,I}$. Second, given an interval K, we respectively denote by l_K and r_K its left and right end points. Third, recall that we denote by F the set of end points of G_S (see Section 4). Given an interval J, we define the *profile* of J with respect to $(S, \rightsquigarrow)$, φ, and I as follows.

$$\mathsf{profile}((S, \rightsquigarrow), \varphi, I, J) = \begin{cases} \mathsf{prefix}(\tau_{\varphi,I}) & \text{if } l_I = l_J \wedge r_I > r_J \\ \mathsf{infix}(\tau_{\varphi,I}) & \text{if } l_I < l_J \wedge r_I > r_J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \in F \setminus J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) \cdot \mathsf{first}(\tau_{\varphi,I'}) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \in F \cap J \\ \tau_{\varphi,I} \cdot \kappa(\varphi, I, J) \cdot \mathsf{prefix}(\tau_{\varphi,I'}) & \text{if } l_I = l_J \wedge r_I \leq r_J \wedge r_J \notin F \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \in F \setminus J \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) \cdot \mathsf{first}(\tau_{\varphi,I'}) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \in F \cap J \\ \mathsf{suffix}(\tau_{\varphi,I}) \cdot \kappa(\varphi, I, J) \cdot \mathsf{prefix}(\tau_{\varphi,I'}) & \text{if } l_I < l_J < r_I \leq r_J \wedge r_J \notin F \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

where we assume J is trimmed to fit the temporal domain of S and $I' \in G_S$ is such that $r_J \in I'$. Moreover, $\kappa(\varphi, I, J)$ is the concatenation $\tau_{\varphi,I_1} \cdot \ldots \cdot \tau_{\varphi,I_m}$ such that $I, I_1, \ldots, I_m, I'$ are consecutive segments in G_S. If $I_1, \ldots, I_m$ do not exist, we let $\kappa(\varphi, I, J) = \{\epsilon\}$. Note that the last case happens when $I \cap J$ is empty.

We now formalize the intuitive approach of "sliding" J over the segmentation to obtain the various profiles it produces as follows.

$$\mathsf{pfs}((S, \rightsquigarrow), \varphi, I, J) = \{\mathsf{destutter}(\mathsf{profile}((S, \rightsquigarrow), \varphi, I, J')) \mid J' \subseteq I \oplus J, J' \sim J\}$$

where $J' \sim J$ holds when $|J'| = |J|$ and J' contains an end point (left or right) iff J does so. Note that although infinitely many intervals J' satisfy the conditions given above (due to denseness of time), the set defined by pfs is finite. We demonstrate this and the computation of pfs in Example 15 and Figure 3.

Example 15. Recall the distributed signal $(S, \rightsquigarrow)$ in Example 8 and its γ function given in Figure 2b. We demonstrate the computation of $\mathsf{pfs}((S, \rightsquigarrow), x_1, [1, 3), [0, 1))$. Intuitively, sliding the interval $[0, 1)$ over the window $[1, 3) \oplus [0, 1)$ (as shown in Figure 3) gives us the following sets:

$$\begin{aligned}
P_1 &= \mathsf{destutter}(\mathsf{prefix}(\gamma(x_1, [1, 3)))) = \{0, 01, 1\} \\
P_2 &= \mathsf{destutter}(\mathsf{infix}(\gamma(x_1, [1, 3)))) = \{0, 01, 1\} \\
P_3 &= \mathsf{destutter}(\mathsf{suffix}(\gamma(x_1, [1, 3)))) = \{0, 01, 1\} \\
P_4 &= \mathsf{destutter}(\mathsf{suffix}(\gamma(x_1, [1, 3))) \cdot \mathsf{prefix}(\gamma(x_1, [3, 4)))) \\
&= \{0, 01, 010, 0101, 01010, 1, 10, 101, 1010\}
\end{aligned}$$

Therefore, we obtain $\mathsf{pfs}((S, \rightsquigarrow), x_1, [1, 3), [0, 1)) = \{P_1, P_2, P_3, P_4\}$. This set over-approximates the potential behaviors of x_1, for all $t \in [1, 3)$, in the interval $t \oplus [0, 1)$.

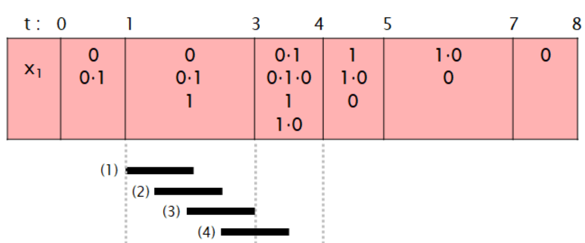

Fig. 3. The profiles of $J = [0, 1)$ with respect to $x_1 \in S$ of Example 8. The four representative intervals of each profile is shown with solid black lines below the tabular representation of γ for x_1.

Let φ_1 and φ_2 be two STL formulas. Intuitively, once we have the profiles of a given interval J with respect to φ_1 and φ_2, we can evaluate the corresponding untimed formulas on the product of these profiles and concatenate them. Formally, we handle the evaluation of timed formulas inductively as follows.

$$[\![(S, \rightsquigarrow), I \models \varphi_1 \mathcal{U}_J \varphi_2]\!] = \mathsf{destutter}(\{u_1 \mathsf{U}^0 u_2 \mid (u_1, u_2) \in P_1 \otimes Q_1\} \cdot \ldots \cdot \{u_1 \mathsf{U}^0 u_2 \mid (u_1, u_2) \in P_k \otimes Q_k\})$$

where $\mathsf{pfs}((S,\rightsquigarrow),\varphi_1,I,J) = \{P_1,\ldots,P_k\}$ and $\mathsf{pfs}((S,\rightsquigarrow),\varphi_2,I,J) = \{Q_1,\ldots,Q_k\}$ such that the intervals producing P_i and Q_i respectively start before those producing P_{i+1} and Q_{i+1} for all $1 \leq i < k$.

Example 16. Let $(S,\rightsquigarrow)$ be as in Example 8 and its γ function as given in Figure 2b. We demonstrate the evaluation of the timed formula $\Diamond_{[0,1)}\, x_1$ over the segment $[1,3)$. Recall from Example 15 the set $\mathsf{pfs}((S,\rightsquigarrow),x_1,[1,3),[0,1)) = \{P_1,P_2,P_3,P_4\}$ of profiles. First, we apply the bitwise eventually operator to each value expression in each of these profiles separately: $\{\mathsf{E}u \mid u \in P_1\} = \{0,1\}$, $\{\mathsf{E}u \mid u \in P_2\} = \{0,1\}$, $\{\mathsf{E}u \mid u \in P_3\} = \{0,1\}$, and $\{\mathsf{E}u \mid u \in P_4\} = \{0,10,1\}$. Then, we concatenate these sets and destutter to obtain the following:

$$[\![(S,\rightsquigarrow),[1,3) \models \Diamond_{[0,1)}\, x_1]\!] = \{0,01,010,0101,01010,1,10,101,1010\}$$

Computing the Semantics of STL$^+$ Putting it all together, given a distributed signal $(S,\rightsquigarrow)$ and an STL$^+$ formula φ, we can compute $[(S,\rightsquigarrow) \models \varphi]_+$ thanks to the following theorem.

Theorem 17. *For every distributed signal $(S,\rightsquigarrow)$, we have $[(S,\rightsquigarrow) \models \varphi]_+ = \top$ (resp. $\bot$, $?$) iff $\mathsf{first}([\![(S,\rightsquigarrow) \models \varphi]\!]) = \{1\}$ (resp. $\{0\}$, $\{0,1\}$).*

Sets of Boolean Value Expressions as Bit Vectors Evidently, asynchronous products are expensive to compute. Our implementation of the algorithm we describe in this section relies on the following observation: Sets of boolean value expressions and their operations can be efficiently implemented through bit vectors. Intuitively, to represent such a set, we can encode each element using its first bit and its length since value expressions are boolean and always destuttered. Moreover, to evaluate untimed operations on such sets, we only need to know the maximal lengths of the four possible types of expressions ($0\ldots0$, $0\ldots1$, $1\ldots0$, and $1\ldots1$) and whether the set contains 0 or 1 (to handle some edge cases). This is because value expressions corresponding to same segments can be seen as completely asynchronous and the possible interleavings obtained from shorter expressions can be obtained from longer ones. This approach enables, for example, an algorithm for conjunction of sets of value expressions that runs in $O(|u|+|v|)$ time where u and v are the longest expressions in the two sets. Note that the same idea also applies to untimed temporal operators.

6 Experimental Evaluation

TODO

7 Conclusion

TODO

References

1. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. Int. J. Softw. Tools Technol. Transf. **15**(3), 247–268 (2013). https://doi.org/10.1007/s10009-012-0247-9
2. Momtaz, A., Abbas, H., Bonakdarpour, B.: Monitoring signal temporal logic in distributed cyber-physical systems. In: Mitra, S., Venkatasubramanian, N., Dubey, A., Feng, L., Ghasemi, M., Sprinkle, J. (eds.) Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems, ICCPS 2023, (with CPS-IoT Week 2023), San Antonio, TX, USA, May 9-12, 2023. pp. 154–165. ACM (2023). https://doi.org/10.1145/3576841.3585937

Appendix

Proof of Theorem 6

Proof. Let φ be an STL formula and $(S, \rightsquigarrow)$ be a distributed signal. Assume $[(S, \rightsquigarrow) \models \varphi]_+ = \top$. We want to show that $[(S, \rightsquigarrow) \models \varphi] = \top$. Expanding the definition of $[(S, \rightsquigarrow) \models \varphi]_+ = \top$, we have $w \models \varphi$ for all $w \in \mathsf{Tr}^+(S, \rightsquigarrow)$. By Lemma 11, we have $\mathsf{Tr}(S, \rightsquigarrow) \subseteq \mathsf{Tr}^+(S, \rightsquigarrow)$. Then, it holds that $w \models \varphi$ for all $w \in \mathsf{Tr}(S, \rightsquigarrow)$. Therefore, $[(S, \rightsquigarrow) \models \varphi] = \top$ by definition. The case of $[(S, \rightsquigarrow) \models \varphi]_+ = \bot$ follows from the same arguments.

Proof of Lemma 11

Proof. Let $(S, \rightsquigarrow)$ be a distributed signal where $S = (x_1, \ldots, x_n)$. Let $w = (y_1, \ldots, y_n) \in \mathsf{Tr}(S, \rightsquigarrow)$ be a trace. We want to show that $w \in \mathsf{Tr}^+(S, \rightsquigarrow)$. First, let us recall the definition of Tr^+.

$$\mathsf{Tr}^+(S, \rightsquigarrow) = \{(x'_1, \ldots, x'_n) \mid x'_i \text{ is consistent with } x_i \text{ for all } 1 \leq i \leq n\}$$

Let $1 \leq i \leq n$ be arbitrary. To show that y_i is consistent with x_i, we need to show that y_i is I-consistent with x_i for all $I \in G_S$. Let $I = [t_0, s)$ be an arbitrary segment in G_S, let $(t_1, y_i(t_1)), \ldots, (t_\ell, y_i(t_\ell))$ be the edges of y_i in segment I with $t_j < t_{j+1}$ for all $1 \leq j < \ell$. To show that y_i is I-consistent with x_i, we need to show that the expression $y_i(t_0) \cdot y_i(t_1) \cdot \ldots \cdot y_i(t_\ell)$ belongs to $\gamma(x_i, I)$. We sketch the proof idea below.

Note that w can be seen as a trace obtained through an ε-retiming of S (see [2, Section 4.2]). Then, the timestamp t of any edge of x_i is mapped to some clock value in the range $(\theta_{\mathrm{lo}}(t), \theta_{\mathrm{hi}}(t))$. In particular, $|t - c_i^{-1}(t)| < \varepsilon$ for all $t \in \{t_0, t_1, \ldots, t_\ell\}$, where $c_i^{-1}(t)$ is the local clock value of x_i that is mapped to t.

Since y_i has ℓ edges in I, it holds that x_i has at least ℓ edges in $(t_0 - \varepsilon, s + \varepsilon)$. Since I is a segment in G_S, there are ℓ of these that are consecutive such that the intersection of their uncertainty regions contain (t_0, s), i.e., $(t_0, s) \subseteq \bigcap_{1 \leq j \leq \ell} (\theta_{\mathrm{lo}}(t'_j), \theta_{\mathrm{hi}}(t'_j))$ where $t'_j = c_i^{-1}(t_j)$ is the corresponding timestamp in x_i for all $0 \leq j \leq \ell$. In particular, note that $y_i(t_j) = x_i(t'_j)$ for all $0 \leq j \leq \ell$.

Now, notice that, by definition, $\gamma(x_i, I)$ takes into account every edge of x_i whose uncertainty region has a nonempty intersection with I, and preserves their order. Let V_j be the set of value expressions capturing how I relates with the uncertainty intervals of the edge $(t'_j, x_i(t'_j))$ for all $1 \leq j \leq \ell$ (as defined in Equation (1)). Then, $\mathsf{destutter}(\{x_i(t'_0)\} \cdot V_1 \cdot \ldots \cdot V_\ell) \subseteq \gamma(x_i, I)$. One can verify that for all $1 \leq j \leq \ell$, either $x_i(t'_j)$ or $x_i(t'_{j-1}) \cdot x_i(t'_j)$ belongs to V_j. This allows us to choose a value expression v_j from each V_j such that $\mathsf{destutter}(\{x_i(t'_0)\} \cdot v_1 \cdot \ldots \cdot v_\ell) = x_i(t'_0) \cdot x_i(t'_1) \cdot \ldots \cdot x_i(t'_\ell)$, which concludes the proof.

Note that if there are more edges of x_i with a timestamp smaller than t'_0 or larger than t'_ℓ whose uncertainty intervals intersect with I, then the corresponding set of value expressions is obtained either by prefixing or suffixing. In either case, we can choose ϵ from these sets for concatenation with the remaining edges' value expressions and obtain the desired result.

Proof of Theorem 17

Proof. TODO