

Approximate Distributed Monitoring under Partial Asynchrony: Balancing Speed with Accuracy

Author(s)

Affiliation(s)

Abstract. In distributed systems with processes that do not share a global clock, *partial asynchrony* is achieved by clock synchronisation that guarantees bounded clock skew among all applications. Existing solutions for distributed runtime verification under partial asynchrony 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 synchronisation is often needed. While perfect clock synchronisation 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 synchronised clocks. In that case, we say that a distributed system has *partial asynchrony*.

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 asynchrony 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 asynchrony. 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 define boolean domain $\mathbb{B} = \{\perp, \top\}$ as the set of boolean truth values, where $\perp < \top$ and they complement each other, i.e., $\overline{\perp} = \top$ and $\overline{\top} = \perp$. We denote by \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 $\text{prefix}(u)$ and $\text{suffix}(u)$ for the sets of prefixes and suffixes of u . We also let $\text{infix}(u) = \{v \in \Sigma^* \mid \exists x, y \in \Sigma^* : u = xvy\}$. For a nonempty word $u \in \Sigma^*$ and $1 \leq i \leq |u|$,

we denote by $u[i]$ the i th 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 $\text{first}(L) = \{u[0] \mid u \in L\}$. For sets $L_1, L_2 \in \Sigma^*$ 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, \dots, u_m) and (v_1, \dots, v_m) of words, we let $(u_1, \dots, u_m) \cdot (v_1, \dots, v_m) = (u_1 v_1, \dots, u_m v_m)$.

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

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

Signal Temporal Logic (STL) Let $A, B \subset \mathbb{R}$. A function $f : A \rightarrow B$ is *right-continuous* iff $\lim_{a \rightarrow c^+} f(a) = f(c)$ 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 $x : [0, d) \rightarrow \mathbb{R}$ where $d \in \mathbb{R}_{>0}$ is the duration of x and $[0, d)$ is its temporal domain. Let $x : [0, d) \rightarrow \mathbb{R}$ 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 $\lim_{s \rightarrow t^-} x(s) \neq \lim_{s \rightarrow t^+} x(s)$. In particular, an edge is *rising* if $\lim_{s \rightarrow t^-} x(s) < \lim_{s \rightarrow t^+} x(s)$, and it is *falling* otherwise. A signal $x : [0, d) \rightarrow \mathbb{R}$ can be represented finitely by its initial value and edges: if x has m edges, then $x = (t_0, v_0)(t_1, v_1) \dots (t_m, v_m)$ 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 $p \in \text{AP}$ 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* $w = (x_1, \dots, x_n)$ is a finite vector of signals. We express atomic propositions as functions of trace values at a time point t , i.e., a proposition $p \in \text{AP}$ over a trace $w = (x_1, \dots, x_n)$ is defined as $f_p(x_1(t), \dots, x_n(t)) > 0$ where $f_p : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 $w = (x_1, \dots, x_n)$ with $x_i : [0, d) \rightarrow \mathbb{R}$ for all $1 \leq i \leq n$. Let φ_1, φ_2 be STL formulas and let $t \in [0, d)$.

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

119 We simply write $w \models \varphi$ for $(w, 0) \models \varphi$. We additionally use the following
120 standard abbreviations: **false** = $p \wedge \neg p$, **true** = $\neg \mathbf{false}$, $\varphi_1 \vee \varphi_2 = \neg(\neg \varphi_1 \wedge$
121 $\neg \varphi_2)$, $\Diamond_I \varphi = \mathbf{true} \mathcal{U}_I \varphi$, and $\Box_I \varphi = \neg \Diamond_I \neg \varphi$. Moreover, the untimed temporal
122 operators are defined through their timed counterparts on the interval $(0, \infty)$,
123 e.g., $\varphi_1 \mathcal{U} \varphi_2 = \varphi_1 \mathcal{U}_{(0, \infty)} \varphi_2$.

124 **Distributed Semantics of STL** We consider an asynchronous and loosely-
125 coupled message-passing system of $n \geq 2$ reliable agents producing a set of
126 signals x_1, \dots, x_n , where for some $d \in \mathbb{R}_{>0}$ we have $x_i : [0, d) \rightarrow \mathbb{R}$ for all
127 $1 \leq i \leq n$. The agents do not share memory or a global clock. Only to formalize
128 statements, we speak of a *hypothetical* global clock and denote its value by T .
129 For local time values, we use the lowercase letters t and s .

130 For a signal x_i , we denote by V_i the set of its events, by E_i^\uparrow the set of its
131 rising edges, and by E_i^\downarrow that of falling edges. Moreover, we let $E_i = E_i^\uparrow \cup E_i^\downarrow$. We
132 represent the local clock of the i th agent as an increasing and divergent function
133 $c_i : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ that maps a global time T to a local time $c_i(T)$. We denote by
134 c_i^{-1} the inverse of the local clock function c_i .

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

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

- 142 – For every agent, the events of its signals are totally ordered, i.e., for all $1 \leq$
143 $i \leq n$ and all $(t, x_i(t)), (t', x_i(t')) \in V_i$, if $t < t'$ then $(t, x_i(t)) \rightsquigarrow (t', x_i(t'))$.
- 144 – Every pair of events whose timestamps are at least ε apart is totally ordered,
145 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 $t + \varepsilon \leq t'$
146 then $(t, x_i(t)) \rightsquigarrow (t', x_j(t'))$.

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

148 **Definition 3.** Let (S, \rightsquigarrow) be a distributed signal of n signals, and $V = \bigcup_{i=1}^n V_i$
149 be the set of its events. A set $C \subseteq V$ is a consistent cut iff for every event in
150 C , all events that happened before it also belong to C , i.e., for all $e, e' \in V$, if
151 $e \in C$ 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* $\text{front}(C) \subseteq C$ is the set consisting of the last events in C of each signal, i.e., $\text{front}(C) = \bigcup_{i=1}^n \{(t, x_i(t)) \in V_i \cap C \mid \forall t' > t : (t', x_i(t')) \notin V_i \cap C\}$.

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

For all $T, T' \in \mathbb{R}_{\geq 0}$ and $1 \leq i \leq n$, if $T < T'$, then for every pair of events $(c_i(T), x_i(c_i(T))) \in \text{ccf}(T)$ and $(c_i(T'), x_i(c_i(T')))) \in \text{ccf}(T')$ we have $(c_i(T), x_i(c_i(T))) \rightsquigarrow (c_i(T'), x_i(c_i(T')))$. We denote by $\text{CCF}(S, \rightsquigarrow)$ 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, \dots, x_n . A consistent cut flow $\text{ccf} \in \text{CCF}(S, \rightsquigarrow)$ yields a trace $w_{\text{ccf}} = (x'_1, \dots, x'_n)$ on a temporal domain $[0, D]$ where $D \in \mathbb{R}_{> 0}$ such that $(c_i(T), x_i(c_i(T))) \in \text{ccf}(T)$ 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 $\text{Tr}(S, \rightsquigarrow) = \{w_{\text{ccf}} \mid \text{ccf} \in \text{CCF}(S, \rightsquigarrow)\}$.

We define the satisfaction of an STL formula φ by a distributed signal (S, \rightsquigarrow) over a three-valued domain $\{\top, \perp, ?\}$. If the set of synchronous traces $\text{Tr}(S, \rightsquigarrow)$ 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 $\text{Tr}(S, \rightsquigarrow)$ 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.

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

3 Overapproximation of the STL Distributed Semantics

To address the computational overhead in exact distributed monitoring, we define a new logic STL^+ whose syntax is the same as STL but semantics provide a sound approximation of the STL distributed semantics presented in Section 2. In essence, given a distributed signal (S, \rightsquigarrow) , STL^+ considers an overapproximation $\text{Tr}^+(S, \rightsquigarrow)$ of the set $\text{Tr}(S, \rightsquigarrow)$ of synchronous traces. A signal (S, \rightsquigarrow) satisfies (resp. violates) an STL^+ formula φ iff all the traces in $\text{Tr}^+(S, \rightsquigarrow)$ belong to the language of φ (resp. $\neg \varphi$).

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

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

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

191 4 Overapproximation of Synchronous Traces

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

199 *Remark 7.* We assume boolean signals in this section for convenience. The defi-
 200 nitions and results presented here extend to real-valued signals.

201 **Canonical Segmentation** Consider a boolean signal x with a rising edge at
 202 time $t > \varepsilon$. Due to clock skew, this edge occurs in the range $(t - \varepsilon, t + \varepsilon)$
 203 from the monitor's point of view. This range is called an *uncertainty region*
 204 because in $(t - \varepsilon, t + \varepsilon)$ the monitor cannot tell the value of x precisely, but
 205 only that it changes from 0 to 1. Formally, given an edge $(t, x(t))$, we define
 206 $\theta_{\text{lo}}(x, t) = \max(0, t - \varepsilon)$ and $\theta_{\text{hi}}(x, t) = t + \varepsilon$ as the end points of the edge's
 207 uncertainty region.

208 Given a temporal domain $I = [0, d) \subset \mathbb{R}_{\geq 0}$, a *segmentation* of I is a partition
 209 of I into finitely many intervals I_1, \dots, I_k , called *segments*, of the form $I_j =$
 210 $[t_j, t_{j+1})$ such that $t_j < t_{j+1}$ for all $1 \leq j \leq k$. By extension, a segmentation of
 211 a collection of signals with the same temporal domain I is a segmentation of I .

212 Let (S, \rightsquigarrow) be a distributed signal of n signals. The *canonical segmentation*
 213 G_S of (S, \rightsquigarrow) is the segmentation of S where the end points of the segments co-
 214 incide with the end points of its temporal domain and uncertainty regions. For-
 215 mally, we define G_S as follows. For each signal x_i , let F_i be the set of end points
 216 of its uncertainty regions. Let $d' = \max(d, \max(\bigcup_{i=1}^n F_i))$, which corresponds to
 217 the duration of the distributed signal with respect to the monitor's clock. Let
 218 $F = \{0, d'\} \cup \bigcup_{i=1}^n F_i$ and let $(s_j)_{1 \leq j \leq |F|}$ be a nondecreasing sequence of clock
 219 values corresponding to the elements of F . Then, the canonical segmentation of
 220 (S, \rightsquigarrow) is $G_S = \{I_1, \dots, I_{|F|-1}\}$ where $I_j = [s_j, s_{j+1})$ for all $1 \leq j < |F|$.

221 *Example 8.* Let (S, \rightsquigarrow) be a distributed boolean signal with $S = (x_1, x_2)$ and
 222 $\varepsilon = 2$ over the temporal domain $[0, 8)$ as given in Figure 1. Both signals are
 223 initially 0. The signal x_1 has a rising edge at time 2 and a falling edge at time 5,
 224 while x_2 has a rising edge at time 3 and a falling edge at time 6. The uncertainty
 225 regions of x_1 are $(0, 4)$ and $(3, 7)$, while those of x_2 are $(1, 5)$ and $(4, 8)$. Then,

226 we have $F = \{0, 8\} \cup \{0, 1, 3, 4, 5, 7, 8\}$, and thus the canonical segmentation is
 227 $G_S = \{[0, 1), [1, 3), [3, 4), [4, 5), [5, 7), [7, 8)\}$.

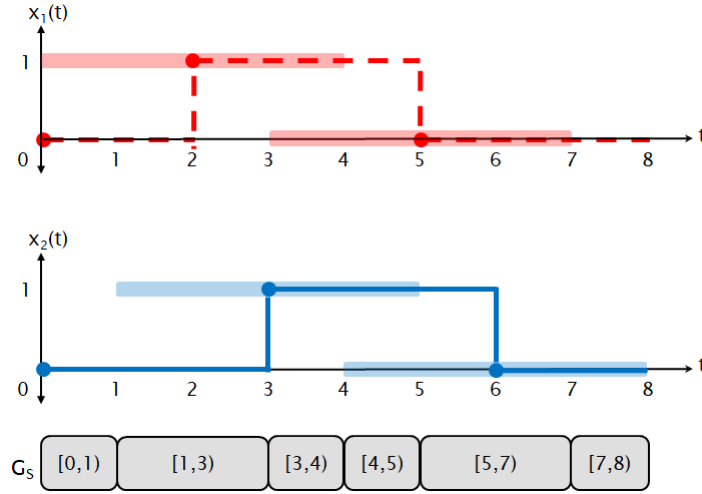


Fig. 1. The signals x_1 (top, red, dashed) and x_2 (bottom, blue, solid) from Example 8. The edges are marked with solid balls and their uncertainty regions are given as semi-transparent boxes around the edges. The resulting canonical segmentation G_S is shown below the graphical representation of the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

228 **Value Expressions** Consider a boolean signal x with a rising edge with an
 229 uncertainty region of (t_1, t_2) . As discussed above, the monitor only knows that
 230 the value of x changes from 0 to 1 in this interval. We represent this knowledge
 231 as a finite word $v = 0 \cdot 1$. This representation is called a *value expression* and
 232 it encodes the uncertain behavior of an observed signal relative to the monitor.
 233 Formally, a value expression is an element of Σ^* where Σ is the finite alphabet
 234 of values the signal takes. Given a signal x and an edge $(t, x(t))$, the value
 235 expression corresponding to the uncertainty region $(\theta_{lo}(x, t), \theta_{hi}(x, t))$ is given
 236 by $v_{x,t} = v_- \cdot v_+$ where $v_- = \lim_{s \rightarrow t^-} x(s)$ and $v_+ = \lim_{s \rightarrow t^+} x(s)$.

237 Notice that (i) uncertainty regions may overlap, and (ii) the canonical seg-
 238 mentation may split an uncertainty region into multiple segments. Consider a
 239 signal x with a rising edge in $(1, 5)$ and a falling edge in $(4, 8)$. The corresponding
 240 value expressions are respectively $v_1 = 0 \cdot 1$ and $v_2 = 1 \cdot 0$. Notice that the be-
 241 havior of x in the interval $[1, 4)$ can be expressed as $\text{prefix}(v_1)$, encoding whether
 242 the rising edge has happened yet or not. Similarly, the behavior in $[4, 5)$ is given
 243 by $\text{suffix}(v_1) \cdot \text{prefix}(v_2)$, 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 $\gamma : S \times G_S \rightarrow 2^{\Sigma^*}$ 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, \dots, R_m be its uncertainty regions where $R_i = (t_i, t'_i)$ and the corresponding value expression is v_i for all $1 \leq i \leq m$. Now, let $I \in G_S$ be a segment with $I = [s, s']$ and for each $1 \leq i \leq m$ define the set V_i of value expressions capturing how I relates with R_i as follows:

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

Note that the last case happens only when $I \cap R_i$ is empty. We finally define $\gamma(x, I) = \text{destutter}(V_1 \cdot V_2 \cdot \dots \cdot V_m) \setminus \{\epsilon\}$. 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, \dots, 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 $\text{suffix}(0 \cdot 1) \cdot \text{prefix}(1 \cdot 0)$ 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 $\text{Tr}^+(S, \rightsquigarrow)$ of synchronous traces that overapproximates the set $\text{Tr}(S, \rightsquigarrow)$.

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 $(t_1, x'(t_1)), \dots, (t_\ell, x'(t_\ell))$ be the edges of x' in segment I with $t_i < t_{i+1}$ for all $1 \leq i < \ell$. The signals x and x' are *consistent in I* iff the value expression $x'(s) \cdot x'(t_1) \cdot \dots \cdot x'(t_\ell)$ belongs to $\gamma(x, I)$. Moreover, x and x' are *consistent* iff they are consistent in I for all $I \in G_S$. Now, let $S = (x_1, \dots, x_n)$ and define $\text{Tr}^+(S, \rightsquigarrow)$ as follows:

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

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

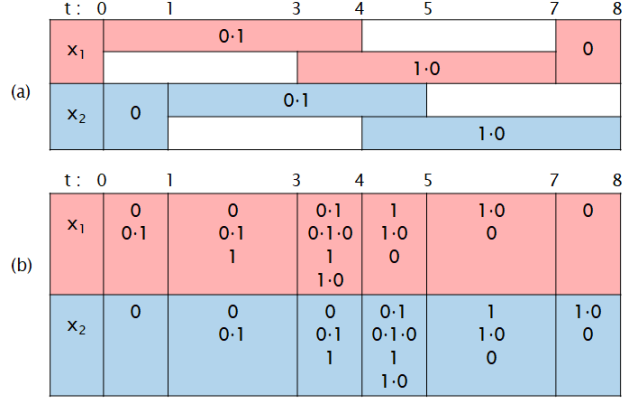


Fig. 2. (a) The uncertainty regions of the distributed signal in Example 8 and the corresponding value expressions. (b) The tabular representation of the function γ for the given distributed signal.

278 contained in $\gamma(x_i, [3, 4))$ and $\gamma(x_i, [4, 5))$ while 0 is contained in the remaining
 279 sets γ maps x_i to.

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

287 Finally, we show the correctness of the canonical overapproximation.

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

289 5 Monitoring Algorithm

290 In this section, given a distributed signal (S, \rightsquigarrow) , we describe an algorithm to
 291 compute $[(S, \rightsquigarrow) \models \varphi]_+$. The algorithm makes use of the function γ defined in
 292 Section 4 without explicitly computing $\text{Tr}^+(S, \rightsquigarrow)$. To achieve this, we first de-
 293 scribe the notion of *asynchronous product* of value expressions to capture poten-
 294 tial interleavings within segments. Then, we present an algorithm to compute
 295 the *semantics* $[(S, \rightsquigarrow) \models \varphi]_+$ using operations on value expressions and their
 296 asynchronous products that encode γ . Finally, we discuss how *bit vectors* can
 297 represent asynchronous products and allow us to manipulate them faster.

298 *Remark 12.* For the sake of convenience, we focus on boolean signals for the rest
 299 of the section. Note that asynchronous products and the algorithm to compute

300 $[(S, \rightsquigarrow) \models \varphi]_+$ can be extended to value expressions over arbitrary finite alpha-
 301 blets, e.g., encoding real-valued signals. This allows us to express more complex
 302 properties where atomic propositions can be functions of real-valued signals.

303 **Asynchronous Product of Value Expressions** Consider the value expres-
 304 sions $u_1 = 0 \cdot 1$ and $u_2 = 1 \cdot 0$ encoding the behaviors of two signals within a
 305 segment. Due to partial asynchrony, the behaviors within segments can be seen
 306 as completely asynchronous. To capture the potential interleavings of these be-
 307 haviors, we consider how the values in u_1 and u_2 can align. In particular, there
 308 are three potential alignments: (i) the rising edge of u_1 happens before the falling
 309 edge of u_2 , (ii) the falling edge of u_2 happens before the rising edge of u_1 , and
 310 (iii) the two edges happen simultaneously. We respectively represent these with
 311 the tuples $(011, 110)$, $(001, 100)$, and $(01, 10)$ where the first component encodes
 312 u_1 and the second u_2 .

Formally, given two value expressions u_1 and u_2 , we define their *asynchronous product* as follows:

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

313 Moreover, given two sets L_1 and L_2 of value expressions, we write $L_1 \otimes L_2 =$
 314 $\{u_1 \otimes u_2 \mid u_i \in L_i, i \in \{1, 2\}\}$.

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

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

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

333 Let us start with the building blocks of our approach. First, we define bitwise
 334 operations on boolean value expressions encoding atomic propositions. Then, we
 335 present how these extend to asynchronous products of value expressions.

336 Let u and v be boolean value expressions of length ℓ . We denote by $u \& v$ the
 337 bitwise-and operation and by $\sim u$ the bitwise-negation operation. In addition, we
 338 define the following bitwise-temporal operators:

TODO: Check these and simplify.

$$\begin{aligned} Eu &= \left(\max_{i \leq j \leq \ell} u[j] \right)_{1 \leq i \leq \ell} & Au &= \left(\min_{i \leq j \leq \ell} u[j] \right)_{1 \leq i \leq \ell} \\ uU^0v &= \left(\max_{i \leq j \leq \ell} \left(\min \left(v[j], \min_{i \leq k \leq j} u[k] \right) \right) \right)_{1 \leq i \leq \ell} \\ uU^1v &= \left(\max \left(u[i..]U^0v[i..], Au[i..] \right) \right)_{1 \leq i \leq \ell} \end{aligned}$$

339 The operators E and A respectively correspond to *eventually* and *always*.
 340 Moreover, U^0 corresponds to the standard (strong) *until* operator while U^1 is
 341 the *weak until*. This distinction will be useful later when we evaluate a formula
 342 segment by segment. We remark that the definitions of these operators coincide
 343 with the robustness semantics of discrete time STL. Finally, note that the output
 344 of these operations is a value expression of length ℓ . For example, if $u = 010$, we
 345 have $Eu = 110$ and $Au = 000$.

346 Now, let L_1 and L_2 be sets of boolean value expressions. We define the
 347 following untimed operations:

TODO: Check these and simplify.

$$\begin{aligned} \neg L_1 &= \{\sim u \mid u \in L_1\} \\ L_1 \cdot L_2 &= \text{destutter}(\{u_1 \& u_2\}) \\ L_1 \wedge L_2 &= \text{destutter}(\{\}) \\ L_1 \mathcal{U} L_2 &= \text{destutter}(\{\}) \end{aligned}$$

348 **EXAMPLES**

349 We additionally define the following timed operators:

350 **TODO**

351 Finally, given a distributed signal (S, \rightsquigarrow) and an STL⁺ formula φ , we compute
 352 $[(S, \rightsquigarrow) \models \varphi]_+$ as follows.

353 **TODO**

354 **Boolean Asynchronous Products as Bit Vectors** getting to atomic propo-
 355 sitions represented as bitvectors intuition on shortcuts
 356 provide equivalent shortcuts for bitvectors.

357 6 Experimental Evaluation

358 **TODO**

359 7 Conclusion

360 **TODO**

361 **References**

- 362 1. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits.
363 Int. J. Softw. Tools Technol. Transf. **15**(3), 247–268 (2013). [https://doi.org/10.](https://doi.org/10.1007/s10009-012-0247-9)
364 1007/s10009-012-0247-9

365 **Appendix**

366 *Proof (Proof of Theorem 6).* **TODO**

367 *Proof (Proof of Lemma 11).* **TODO**

368 *Proof (Proof of Lemma 14).* **TODO**