Monitoring Signal Temporal Logic in Distributed Cyber-physical Systems

Anik Momtaz
momtazan@msu.edu
Michigan State University
East Lansing, Michigan, U.S.A.

Houssam Abbas
houssam.abbas@oregonstate.edu
Oregon State University
Corvallis, Oregon, U.S.A.

Borzoo Bonakdarpour
borzoo@msu.edu
Michigan State University
East Lansing, Michigan, U.S.A.

ABSTRACT

This paper solves the problem of runtime verification for *signal temporal logic* in *distributed* cyber-physical systems (CPS). We assume a partially synchronous setting, where a clock synchronization algorithm guarantees a bound on clock drifts among all signals. We introduce a formula progression and a signal *retiming* technique that allow reasoning about the correctness of formulas among continuous-time and continuous-valued signals that do not share a global view of time. The resulting problem is encoded as a *satisfiability modulo theory* (SMT) solving problem, and we introduce techniques to solve the SMT encoding efficiently. We also conduct two case studies on monitoring a network of aerial vehicles and a water distribution system.

CCS CONCEPTS

• **Theory of computation → Automated reasoning**; • **Computer systems organization → Embedded and cyber-physical systems**; • **Computing methodologies → Distributed computing methodologies**.

KEYWORDS

Runtime verification, Distributed cyber-physical systems, Formal methods

ACM Reference Format:
Anik Momtaz, Houssam Abbas, and Borzoo Bonakdarpour. 2023. Monitoring Signal Temporal Logic in Distributed Cyber-physical Systems. In *ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023) (ICCPS '23), May 9–12, 2023, San Antonio, TX, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3576841.3585937

1 INTRODUCTION

Cyber-physical systems (CPS) are making their way in masses into our environment. With the advent of the Internet of Things (IoT) and edge applications, CPS are particularly becoming *distributed* over networks of *agents*. Examples of such *multi-agent* CPS include networks of sensors in infrastructures, health-monitoring wearables, networks of medical devices, and autonomous vehicles. CPS generally have a safety-critical nature, hence, gaining assurance about their correctness is crucial. One way to gain such assurance is by *monitoring* distributed CPS with respect to their formal specification in a systematic way. While there have been proposals for monitoring temporal logics for distributed discrete-event systems (e.g., [12, 20, 22]), we currently lack a universal theory that allows for monitoring distributed CPS with respect to sophisticated specification languages such as *signal temporal logic* (STL).

The challenge of monitoring distributed CPS stems from two key characteristics. First, in CPS, signals are *analog* and consist of an infinite set of events which makes existing reasoning techniques from the discrete-event distributed settings inapplicable. Second, agents in distributed CPS often have independent *local clocks* that may drift from each other over time. In fact, it is unclear when exactly an infinity of events are sequential or concurrent. Thus, the notion of *global* time in centralized CPS does not hold anymore.

The above characteristics constitute the notion we call *distributed signals*, where reasoning about them demands establishing some concept of serialization. However, building such serializations for an infinite number of events from multiple signals under clock drifts at run time is currently an uncharted territory. For instance, consider the water distribution system shown in Figure 1, where multiple water tanks (located at different offsite water distribution centers) supply water to a critical system in the event of an emergency. The outflow rate and pressure of these water tanks are measured locally using local clocks that are subject to clock drift. Now, if the compounded pressure and flow rate of these tanks are to be monitored, then a monitor must observe the values reported by these tanks that happen to be timestamped by their respective drifting local clocks. As the monitor only has access to the continuous signals representing the pressure and flow rate of these tanks that are (at best) partially synchronized, accurately assessing these values becomes a challenging task indeed. If the flow rate and pressure have to remain under a certain threshold at all times, then due to clock drift amongst the local clocks, it is possible to miss values for which the threshold is breached.

In [19], the authors propose a technique for monitoring Boolean predicates over a set of distributed signals. Our work in this paper builds on [19] by extending monitoring Boolean predicates over distributed signals to the full signal temporal logic (STL) [10]. To this end, we first assume a *partially synchronous* setting, where a clock synchronization algorithm guarantees a maximum bound ε on clock drifts among all signals. This can be ensured by off-the-shelf algorithms such as NTP [17]. We incorporate the notion of signal *retiming* introduced in [19] that allows aligning continuous-time signals that do not share a global view of time. Assuming the bound ε, the decision problem is to search for a retiming function that results in violation of an input STL formula. If such a function does

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICCPS '23, May 9–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0036-1/23/05…$15.00
https://doi.org/10.1145/3576841.3585937

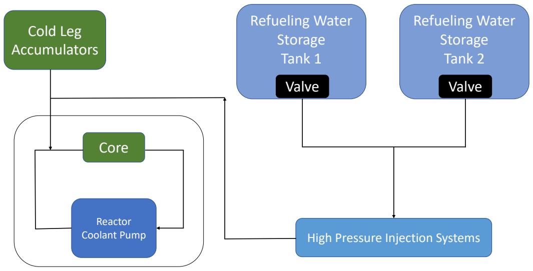

Cold Leg Accumulators
Refueling Water Storage Tank 1
Valve
Refueling Water Storage Tank 2
Valve
Core
Reactor Coolant Pump
High Pressure Injection Systems

Figure 1: Hybrid dynamic cooling system with water tanks.

not exist, it implies that the distributed signals have not yet violated the formula (it may or may not in the future).

In order to cope with the computational complexity, given a distributed signal, we split the original signal into smaller signals that we refer to as *segments*. The challenge here is that the result of monitoring one segment should carry over to the next segment. For example, consider STL formula $\varphi = \Box_{[0,5]}\, p$ (which means proposition p should hold at all times in time interval $[0, 5]$) and the current segment of signals that end at time 3. This means if p holds in the interval $[0, 3]$, then the formula has to be rewritten to $\varphi' = \Box_{[0,2]}\, p$ for the second segment. Of course, such rewriting can become challenging when the formulas have multiple nested temporal operators with relative time intervals. To this end, we propose a *formula progression* technique that takes as inputs an STL formula and a finite-time distributed signal σ and returns an STL formula φ' such that for any extension σ', we have $\sigma\sigma' \models \varphi$ if and only if $\sigma' \models \varphi'$. We encode the resulting problem as a *satisfiability modulo theory* (SMT) problem that searches for a retiming function given the constraints of the current segment and STL formula. We introduce techniques to solve the SMT encoding efficiently.

We have fully implemented our approach on two distributed CPS applications: monitoring of a (1) network of aerial vehicles with respect to a set of properties such as mutual separation and formation, and (2) a water distribution system with respect to the property in which the outflow pressure reaches above the threshold pressure. The results show that in some cases, it is even possible to monitor a distributed CPS sufficiently fast for online deployment.

Organization. In Section 2, we present the background concepts on STL and distributed signals. Section 3 formally states the monitoring problem. Our formula progression technique for STL and the SMT formulation are introduced in Sections 4 and 5, respectively. We evaluate our approach through case studies and experiments in Section 6. Finally, we make concluding results and discuss future work in Section 7. All proofs appear in the appendix.

1.1 Related Work

There is extensive literature on runtime monitoring of CPS where perfect time synchrony is assumed. In [9] the authors propose techniques for online monitoring of STL [10]. A predictive monitoring approach is explored in [8] that requires knowledge on the full system model. In [13] monitoring is done by assuming worst-case bounds on signal values. The effects of timing inaccuracies on satisfaction of temporal properties is studied in [1, 25]. A minimally intrusive CPS monitoring approach is studied in [16].

In [5, 18] approaches for Lattice-theoretic centralized and decentralized online predicate detection in *discreet-event* distributed systems are reported. This is extended to include temporal operators in [20, 22]. A method for monitoring safety properties in distributed systems using the past-time LTL is proposed in [26]. Finally, runtime monitoring of LTL formulas for synchronous distributed systems has been studied in [2, 6, 7].

In the context of monitoring partially synchronous systems, in [29] the feasibility of monitoring partially synchronous distributed systems in order to detect latent bugs was first investigated. This technique was later generalized to full LTL in [12], where the presence of latent bugs are detected using SMT solvers in a discrete setting. A tool for identifying data races in distributed system traces is introduced in [24] for handling non-deterministic discrete event orderings. However, these approaches cannot fully capture the continuous-time and continuous-valued behavior of CPS. Our work is closer to [19] where the authors propose a technique to monitor predicates on a partially synchronous distributed system by retiming continuous signals. While this approach improves monitoring efficiency by levering knowledge about system dynamics, it is limited to only being able to monitor predicates, and cannot capture temporal behavior.

2 PRELIMINARIES

We denote the set of reals as $\mathbb{R}$, the set of non-negative reals as $\mathbb{R}_+$, and the set of positive reals as $\mathbb{R}_+^*$. The set of natural numbers $\{1, \ldots, N\}$ is abbreviated as $[N]$. *Global* (hypothetical) time values are denoted by χ, χ', etc, while t, t', t_1, t_2, s, s', s_1, s_2, etc, denote *local* clock values specific to given signals/agents.

2.1 Signal Model

Definition 2.1. An *output signal* (of some agent A) is a function $x : [a, b] \to \mathbb{R}^d$, which is right-continuous, left-limited, and is not Zeno. Here, $[a, b]$ is an interval in $\mathbb{R}_+$, and will be referred to as the *timeline* of the signal. ■

Without loss of generality, we assume that x is one-dimensional, i.e., $d = 1$. *Right-continuity* means at all t in its support, $\lim_{s\to t_+} x(s) = x(t)$. *Left-limitedness* means the function has a finite left-limit at every t in its support: $\lim_{s\to t_-} x(s) < \infty$. *Not being Zeno* means that x has a finite number of discontinuities in any bounded interval in its support. A *discontinuity* in a signal $x(\cdot)$ can be due to a discrete event internal to agent A (like a variable updated by software).

We consider a loosely coupled system consisting of N *agents* that do not fail, denoted by $\{A_1, \ldots, A_N\}$, without any shared memory or global clock. The output signal of agent A_n is denoted by x_n, for $n \in [N]$. We refer to some global clock which acts as a '*real*' time-keeper. However, this global clock is a hypothetical object used in definitions and theorems. We make the following assumption:

Partial synchrony. The *local clock* (or time) of an agent A_n can be represented as an increasing function $c_n : \mathbb{R}_+ \to \mathbb{R}_+$, where $c_n(\chi)$ is the value of the local clock at global time χ. Then, for any two agents A_n and A_m, where $m, n \in [N]$, we have $\forall \chi \in \mathbb{R}_+ . |c_n(\chi) - c_m(\chi)| < \varepsilon$, where $\varepsilon \in \mathbb{R}_+^*$ is the maximum *clock skew*, that is assumed fixed and known by the monitor. In the sequel, we make it explicit when we refer to 'local' or 'global' time. This assumption is met by using a clock synchronization algorithm, such as NTP [17], to ensure bounded clock skew among all agents.

In the discrete-time setting, an event is a value change in an agent's variables. We now update this definition for the continuous-time setting. Specifically, in an agent A_n, an *event* is a pair $(t, x_n(t))$, where t is the local time (i.e., returned by function c_n defined above).

A *distributed signal* is modeled as a set of signals, where events in each signal are partially ordered by a variation of the *happened-before* ($\rightsquigarrow$) relation [14], extended by our assumption (A1) on bounded clock skew among all agents. The following defines a continuous-time/value *distributed signal* under partial synchrony.

Definition 2.2. A *distributed signal* on N agents is a pair $(E, \rightsquigarrow)$, where $E = (x_1, \dots, x_N)$ is a vector of signals and $\rightsquigarrow$ is a relation between events in signals such that for any bounded non-empty interval I_n:

(1) In any signal x_n, all events are totally ordered, that is, for all $n \in [N]$, for any $t, t' \in I_n$, if $t < t'$, then $(t, x_n(t)) \rightsquigarrow (t', x_n(t'))$.
(2) If the time between any two events is more than the maximum clock skew ε, then the events are totally ordered, that is, for all $m, n \in [N]$, for any $t \in I_n$ and $t' \in I_m$, if $t + \varepsilon < t'$, then $(t, x_n(t)) \rightsquigarrow (t', x_m(t'))$. ■

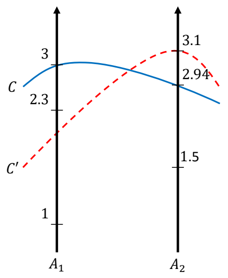


Figure 2: Two partially synchronous concurrent timelines with $\varepsilon = 0.1$.

In Definition, 2.2, the classic case of complete asynchrony is achieved by setting $\varepsilon = \infty$. The restrictions on I_n (bounded and non-empty) are necessary in the continuous-time setting and will be revisited in the next section. As the agents are synchronized within ε, it is not possible to evaluate all signals at the same moment in global time. The notion of *consistent cut*, defined next, captures *possible* global states. We borrow the notion of consistent cuts from [4], and modify it to fit continuous signals. Figure 2 shows two partially synchronous concurrent timelines generated by two agents. Every moment in each timeline corresponds to an event $(t, x_n(t))$, $n \in [2]$. Thus, the following hold: $(1, x_1(1)) \rightsquigarrow (2.3, x_1(2.3))$, $(2.3, x_1(2.3)) \rightsquigarrow (2.94, x_2(2.94))$, $(1.5, x_2(1.5)) \rightsquigarrow (2.94, x_2(2.94))$, and $(2.94, x_2(2.94)) \not\rightsquigarrow (3, x_1(3))$.

Definition 2.3. Let $(E, \rightsquigarrow)$ be a distributed signal over N agents, and $S = \{(t, x_n(t)) \mid x_n \in E \wedge t \in I_n \wedge I_n \subseteq \mathbb{R}_+\}$ be the set of all events. A set $C \subseteq S$ is a *consistent cut* if and only if when C contains an event e, then it contains all events that happened before e. Formally, $\forall e, f \in S \, . \, (e \in C) \wedge (f \rightsquigarrow e) \Rightarrow (f \in C)$. ■

From this definition and Definition 2.2, it follows that if $(t', x_n(t'))$ is in C, then C also contains every event $(t, x_m(t))$ s.t. $t + \varepsilon < t'$. Due to time asynchrony, at any global time $\chi \in \mathbb{R}_+$, there exists infinite number of consistent cuts denoted by $\mathcal{C}(\chi)$. This is due to the existence of an infinite number of time instances between any two local time instances t_1 and t_2 on some signal x. Therefore, an infinite number of consistent cuts can be constructed. A consistent cut C can be represented by its *frontier*:

$$\mathsf{front}(C) = \{(t_1, x_1(t_1)), \dots, (t_N, x_N(t_N))\},$$

in which each $(t_n, x_n(t_n))$, where $1 \leq n \leq N$, is the last event of agent A_n appearing in C. Formally, $\forall n \in [N] \, . \, (t_n, x_n(t_n)) \in C$ and $t_n = \max\{t \in I_n \mid \exists(t, x_n(t)) \in C\}$. In Fig. 2 where $\varepsilon = 0.1$, all events below the solid arc form a consistent cut C with frontier $\mathsf{front}(C) = \{(3, x_1(3)), (2.94, x_2(2.94))\}$. On the other hand, all events below the dashed arc do *not* form a consistent cut since $(2.3, x_1(2.3)) \rightsquigarrow (3.1, x_2(3.1))$ and $(3.1, x_2(3.1))$ is in the set C', but $(2.3, x_1(2.3))$ is not in C'.

2.2 Signal Temporal Logic (STL) [15]

Let AP be a set of *atomic propositions*. We assume *signal temporal logic* (STL) formula to be in the negation normal form. The syntax for STL is defined for infinite signals using the following grammar:

$$\varphi := p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \ \mathcal{U}_{[a,b]} \ \varphi \mid \varphi \ \mathcal{R}_{[a,b]} \ \varphi$$

where $p \in$ AP, and $\mathcal{U}$ (resp., $\mathcal{R}$) is the 'Until' (resp., 'Release') temporal operator. We view other propositional and temporal operators as abbreviations, that is, $\top = p \vee \neg p$ (`true`), $\bot = p \wedge \neg p$ (`false`), $\Diamond_{[a,b]} \varphi = \top \, \mathcal{U}_{[a,b]} \varphi$ (*eventually*), $\Box_{[a,b]} \varphi = \bot \, \mathcal{R}_{[a,b]} \varphi$ (*always*). We denote the set of all STL formulas by Φ_{STL}.

Let a *trace* $\sigma = (x_1, \dots, x_N)$ be a vector of N continuous-time and continuous-valued signals. In the context of STL, we express p as $f(x_1[t], \dots, x_n[t]) > 0$, where $(x_1[t], \dots, x_n[t]) \in \mathbb{R}^n$ is a vector of signal values at time t, and $f : \mathbb{R}^n \to \mathbb{R}$ is a function that evaluates a vector of signal values.

The infinite-trace semantics of STL is defined as follows. Let $\models$ be the *satisfaction* relation, and the satisfaction of formula φ by a trace σ at time t be:

$$\begin{array}{lll}
(\sigma, t) \models p & \text{iff} & f(x_1[t], \dots, x_n[t]) > 0 \\
(\sigma, t) \models \neg p & \text{iff} & f(x_1[t], \dots, x_n[t]) \leq 0 \\
(\sigma, t) \models \varphi \wedge \psi & \text{iff} & (\sigma, t) \models \varphi \text{ and } (\sigma, t) \models \psi \\
(\sigma, t) \models \varphi \vee \psi & \text{iff} & (\sigma, t) \models \varphi \text{ or } (\sigma, t) \models \psi \\
(\sigma, t) \models \varphi \, \mathcal{U}_{[a,b]} \psi & \text{iff} & \exists t' \in [t+a, t+b] : (\sigma, t') \models \psi \text{ and} \\
 & & \forall t'' \in [t, t') : (\sigma, t) \models \varphi \\
(\sigma, t) \models \varphi \, \mathcal{R}_{[a,b]} \psi & \text{iff} & \exists t' \in [t+a, t+b] : (\sigma, t') \models \varphi \text{ and} \\
 & & \forall t'' \in [t, t') : (\sigma, t'') \models \psi
\end{array}$$

Also, $\sigma \models \varphi$ holds if and only if $(\sigma, 0) \models \varphi$ holds. For example, given the trace σ shown in Fig. 3, the STL formula $\varphi = p \, \mathcal{U}_{[4,6.5]} q$ holds at time 0, that is, $\sigma \models \varphi$. However, φ does not hold after time 2, as in that case, q must hold after time $2 + 4$ and before $2 + 6.5$, which does not happen.

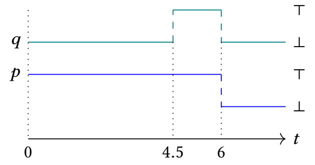


Figure 3: A trace σ generated by a system.

A subtlety is that a distributed signal E is defined to be of finite duration (I_n are bounded), which suits the online monitoring setup, while the STL semantics are over infinite signals. This is handled in the classical manner: given a (fully synchronous) finite duration signal x, we say it satisfies/violates φ iff every extension $(x.y)$, where y is an infinite signal, satisfies/violates φ. Otherwise, the monitor returns Unknown. Here, the dot '.' denotes concatenation in time.

Communication between nodes necessarily involves sampling the analog signal, transmitting the samples, and reconstructing the signal at the receiving node. Our objective is to monitor the *reconstructed analog signals*. This is different from monitoring a discrete-time signal consisting of the samples – the applications we target are concerned with the value of the signal between samples, and potential violations they reveal. Methods for signal transmission, including sampling and reconstruction, are standard in communication theory. Errors due to sampling and reconstruction (say, because of bandwidth limitations) can be accounted for by strictifying the STL formula using the methods of [11]. The choice of reconstruction algorithm is application-dependent and follows domain knowledge. In this paper, for simplicity and without loss of generality, we assume that every output signal x_n is reconstructed as *piece-wise linear* between the samples. This ensures violations can be detected from the piece-wise linear interpolations even if no violations are detected on the samples.

3 PROBLEM STATEMENT

As distributed agents are partially synchronized within ε clock skew, a monitoring algorithm must explore all (infinite) possible reachable consistent cuts. We call the progress of consistent cuts over time a *consistent cut flow*. Our objective is to determine whether there exists some flow of moments that are within ε of each other for which at least one reachable consistent cut results in violation of a given STL formula. This intuition is formalized below, starting with the notion of a *consistent cut flow*.

Definition 3.1 (Consistent cut flow). Let $(E, \rightsquigarrow)$ be a distributed signal over N agents with time interval $[a, b]$, and S be the set of all events over E. A *consistent cut flow* is a function $\mathrm{ccf} : [a, b] \to 2^S$ that maps each time $\chi \in [a, b]$ to the frontier of a consistent cut at time χ; i.e., $\mathrm{ccf}(\chi) \in \{\mathrm{front}(C) \mid C \in \mathcal{C}(\chi)\}$. For each time $\chi' \in [a, b]$, and for each $n \in [N]$, if $\chi < \chi'$, then for all events $(c_n(\chi), x_n(c_n(\chi))) \in \mathrm{ccf}(\chi)$, and for all events $(c_n(\chi'), x_n(c_n(\chi'))) \in \mathrm{ccf}(\chi')$, $(c_n(\chi), x_n(c_n(\chi))) \rightsquigarrow (c_n(\chi'), x_n(c_n(\chi')))$ hold. ■

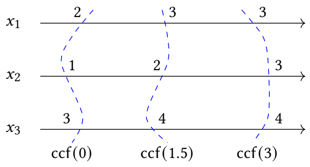


Figure 4: A valid ccf **for the formula** $G_{[0,3]}(x_1 + x_2 + x_3 \leq 10)$**.**

Notice that a consistent cut flow induces a vector of N signals that are fully synchronized, and thus, can be verified against an STL formula φ at time t as $(\mathrm{ccf}, t) \models \varphi$ using the standard semantics described in Section 2.2. That is, for a consistent cut flow ccf on $(E, \rightsquigarrow)$, individual signals $(x'_1, \ldots, x'_N)$ can be constructed, such that, for all $1 \leq i \leq N$ and for all $\chi \in [a, b]$, if $(c_i(\chi), x_i(c_i(\chi))) \in \mathrm{ccf}(\chi)$, then $x'_i(\chi) = x_i(c_i(\chi))$. For example, let $(E, \rightsquigarrow)$ be a distributed signal consisting of signals x_1, x_2, and x_3 as shown Fig. 4. For an STL formula $\square_{[0,3]}(x_1 + x_2 + x_3 \leq 10)$, ccf is a valid consistent flow on $(E, \rightsquigarrow)$. Note that a distributed signal $(E, \rightsquigarrow)$ encodes uncountably infinite consistent cut flows. Let us denote the set of all consistent cut flows by $\mathcal{CCF}$. Our decision problem consists of determining whether there is a violation of a given STL formula by some consistent cut flow.

Definition 3.2 (Distributed satisfaction). Let φ be an STL formula, $(E, \rightsquigarrow)$ be a distributed signal over N agents and $\mathcal{CCF}$ be the set of all induced consistent cut flows. We say that $((E, \rightsquigarrow), 0)$, or simply $(E, \rightsquigarrow)$ *satisfies* φ, iff for each $\sigma \in \mathcal{CCF}$, we have $\sigma \models \varphi$. ■

Problem Statement

Given maximum clock skew $\varepsilon > 0$, a distributed signal $(E, \rightsquigarrow)$ over N agents, and an STL formula φ, decide whether there exists a consistent cut flow $\sigma \in \mathcal{CCF}$ where $\sigma \not\models \varphi$.

4 MONITORING ALGORITHM SKETCH

In this paper, we assume the monitor receives output signals from x_n as *piece-wise linear* signals (this is by choice and other forms of discretization will not change the core monitoring algorithm). This transmission happens in *segments* of length T: at the k^{th} transmission, agent A_n transmits $x_n|_{[(k-1)T,kT]}$, the restriction of its output signal to the interval $[(k-1)T, kT]$ as measured by its local clock. In the rest of this paper, we refer exclusively to the signal fragments received by the monitor in a given transmission.

We now revisit the restriction placed on I_n in Definition 2.2, namely, that the monitor only deals with non-empty bounded signal fragments $x_n|_{[(k-1)T,kT]}$, therefore, $I_n = [(k-1)T, kT]$ for every agent at the k^{th} transmission, *measured in local time*. By the bounded skew assumption, we have:

Lemma 4.1 (Bounded skew lemma). *For any two agents A_n, A_m with intervals $I_n = [\min I_n, \max I_n]$ and $I_m = [\min I_m, \max I_m]$, $|\min I_n - \min I_m| \leq \varepsilon$ and $|\max I_n - \max I_m| \leq \varepsilon$.* ■

Since online monitoring happens in segments, at the end of each segment the monitor either returns $\top$ (formula already satisfied), $\bot$ (already violated), or *unknown*, and the next segment is processed. For simplicity, our solution employs a *central monitor*. Our monitoring algorithm involves three key ideas: (1) formula progression, (2) signal retiming, and (3) SMT-based implementation, explained in Sections 4.1 – 4.3, respectively.

4.1 Formula Progression

Let φ be an STL formula and $(E, \rightsquigarrow)$ be a distributed signal. Without loss of generality, let this signal be split into two segments: prefix $(E_1, \rightsquigarrow)$ and suffix $(E_2, \rightsquigarrow)$, that is, $(E, \rightsquigarrow) = (E_1 E_2, \rightsquigarrow)$. Thus, the monitor first evaluates φ on $(E_1, \rightsquigarrow)$. If the verdict yields true or false, then this verdict is returned and monitoring for $(E, \rightsquigarrow)$ is already complete. Otherwise, the monitor computes a new *progressed formula* φ' which will be evaluated for segment $(E_2, \rightsquigarrow)$.

Definition 4.2 (Formula progression). Let $(E_1, \rightsquigarrow)$ be a finite distributed signal starting at time 0 whose duration is denoted by $|(E_1, \rightsquigarrow)|$, and $(E_2, \rightsquigarrow)$ be a finite or infinite extension of $(E_1, \rightsquigarrow)$. We say STL formula φ' is a *progression* of STL formula φ for $(E_1, \rightsquigarrow)$ if and only if $((E_1 E_2, \rightsquigarrow), 0) \models \varphi \Leftrightarrow ((E_2, \rightsquigarrow), 0) \models \varphi'$. ■

It stands to reason that if $((E_1, \rightsquigarrow), 0) \models \varphi$ (resp., $((E_1, \rightsquigarrow), 0) \not\models \varphi$), then the progression of φ is trivially $\varphi' = \top$ (resp., $\varphi' = \bot$).

4.2 Signal Retiming

Recall that signals are measured using their local clocks. Since the signals in our setting are partially synchronized within an ε, it is not possible to evaluate all signals at the same moment in global time. Rather, the best a monitor can do is explore all valid alignments of the concurrent local moments (i.e., those moments that are within ε of each other) and determine whether at least one such alignment violates the formula. This intuition is formalized below, starting with the notion of a *retiming function* borrowed from [19] that establishes the happened-before relation in the continuous-time setting, and stretches or compresses signals to align them with each other within the ε clock skew bound.

Definition 4.3 (Retiming functions). A *retiming* function (or retiming) is an increasing function $\rho : \mathbb{R}_+ \longrightarrow \mathbb{R}_+$. An ε-retiming is a retiming where $\forall t \in \mathbb{R}_+ : |t - \rho(t)| < \varepsilon$. Given a distributed signal $(E, \rightsquigarrow)$ over N agents and two agents A_i, A_j, where $i, j \in [N]$, a retiming ρ from A_j to A_i *respects* $\rightsquigarrow$ if we have $((t, x_i(t)) \rightsquigarrow (t', x_j(t')))$, then $(t < \rho(t'))$ for any two events $(t, x_i(t)), (t', x_j(t')) \in E$. An ε-retiming that respects $\rightsquigarrow$ is a *valid* retiming. ■

A valid retiming formalizes the notion of alignment of timelines: given two ε-synchronous timelines t and s (on two agents), we treat moments t and $s = \rho(t)$ as being simultaneous. Thus, the signal $x(t) = [x_1(t), x_2(\rho(t))]$ is now a fully synchronous signal. An ε-retiming ρ maps $\mathbb{R}_+$ to itself, but the restriction of ρ to a bounded interval I is an increasing function from I to $\rho(I)$ that respects the constraint $|t - \rho(t)| < \varepsilon$ for all $t \in I$. Thus, we restrict our attention to ε-retimings on bounded intervals. Between 2 agents, we need one retiming $\rho : I_2 \longrightarrow I_1$, and between N agents, we need $N - 1$ retimings $I_n \longrightarrow I_1$. In general there is an infinity of valid retimings, any of which might reveal a potential violation. The next theorem establish the fundamental condition on ε-retimings among agents and violation of an STL formula.

THEOREM 4.4. *Given a distributed signal $(E, \rightsquigarrow)$ over N agents, and an STL formula φ with time interval $[a, b]$, there exists a violation at time $t \in \mathbb{R}_+$, if and only if there exists $N - 1$ ε-retimings $\rho_n : I_n \longrightarrow I_1$ that respect $\rightsquigarrow$, where $2 \leq n \leq N$, such that:*

$$\Big(\big(x_1, x_2 \circ \rho_2^{-1}, \ldots, x_N \circ \rho_N^{-1}\big), t\Big) \not\models \varphi \tag{1}$$

Here, $\rho_m^{-1} \circ \rho_n : I_n \longrightarrow I_m$ is an ε-retiming for all $n \neq m$, and '$\circ$' denotes the function composition operator, where given two functions f and g, $h = g \circ f$ such that $h(x) = g(f(x))$. ■

4.3 SMT Encoding

We solve the monitoring problem by transforming it into an instance of the *satisfiability modulo theory* (SMT). Specifically, we ask whether there exists $N - 1$ retimings, such that (1) holds; equivalently, whether there exists a consistent cut flow that witnesses satisfaction of $\neg\varphi$. That is, the distributed signal violates φ iff the following SMT problem is satisfiable. This transformation to SMT solving is the focus of the next section.

5 SMT-BASED MONITORING ALGORITHM

The SMT formulation part of our solution is constructed by encoding both formula progression and signal retiming into a single SMT-solving problem, and then solving it using an SMT-solver. First, we define the SMT entities and constraints, then demonstrate our monitoring approach with two complete examples. In both examples, we consider a distributed signal $(E, \rightsquigarrow)$ comprised of two individual 10 time unit long signals x_1 and x_2, generated by agents A_1 and A_2 respectively, with a clock skew bound $\varepsilon = 1$. Our running examples involve monitoring formulas $\neg\varphi_1 = \Diamond_{[0,10]}\, p$ and $\neg\varphi_2 = \Diamond_{[0,10]}(p \wedge \Box_{[0,5]}\, \neg q)$.

5.1 SMT Entities

In our encoding, N signals and time intervals are defined in the same fashion as the mathematical representation in previous sections. We also include ρ_n retiming functions, where $2 \leq n \leq N$, a consistent cut flow function ccf as an *uninterpreted* function, and real numbers t, s, and χ. Identifying interpretations of these functions will be the output SMT solving and, hence, the verdict of monitoring. The sampled signal values are constants in the encoding that are known to the monitor: $\{x_n(t_n) \mid t_n \in I_n\}$.

5.2 SMT Constraints – Single Segment

Recall from Section 3 that $(\mathrm{ccf}, t) \models p$ denotes a consistent cut flow at time t on signals $(x'_1, \ldots, x'_N)$ satisfies the atom p. To express this as an SMT problem, we encode $(\mathrm{ccf}, t) \models p$ as $f(x'_1[t], \ldots, x'_n[t]) > 0$, where $(x_1[t], \ldots, x_n[t]) \in \mathbb{R}^n$ is a vector of signal values at time t, and $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a function that evaluates a vector of signal values. The SMT constraints are primarily comprised of (1) a set of constraints that ensures valid consistent cut flow, (2) a set of constraints that find violation, and (3) a set of constraints that enforce valid retimings under a given clock skew.

Consistent cut flow constraints. In order to ensure that ccf identifies a valid consistent cut flow on $(E, \rightsquigarrow)$ over time interval $[a, b]$, first we define the happened-before ($\rightsquigarrow$) notation in SMT according to Definition 2.2, and ensure that the events in the consistent cuts mapped by ccf respect the happened-before relation:

$$\begin{aligned} SMT_{\mathrm{flow}_1} = &\forall \chi \in [a, b]\ .\ \forall (t_n, x_n(t_n)), (t'_n, x_n(t'_n)) \in E\ . \\ &\Big(\big((t'_n, x_n(t'_n)) \rightsquigarrow (t_n, x_n(t_n))\big) \wedge \big((t_n, x_n(t_n)) \in \mathrm{ccf}(\chi)\big)\Big) \\ &\Rightarrow \Big((t'_n, x_n(t'_n)) \in \mathrm{ccf}(\chi)\Big). \end{aligned}$$

And that the consistent cuts mapped by ccf always increase and never intersect:

$$SMT_{\mathrm{flow}_2} = \forall \chi, \chi' \in [a, b]\ .\ \forall n \in [N]\ .\ \Big(\chi < \chi' \Rightarrow c_n(\chi) < c_n(\chi')\Big).$$

Thus, the SMT constraint for consistent cut flow is the following:

$$\boxed{SMT_{\mathrm{flow}} = SMT_{\mathrm{flow}_1} \ \wedge\ SMT_{\mathrm{flow}_2}.}$$

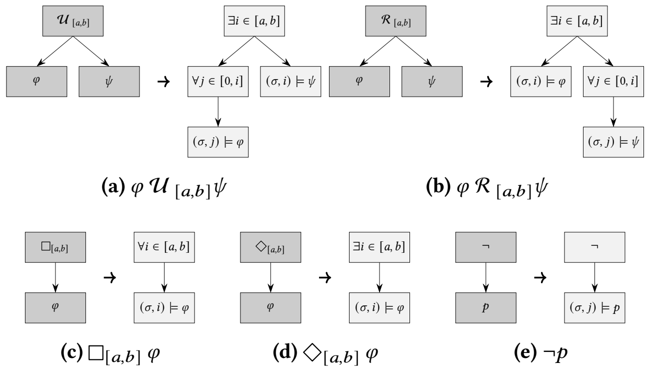
Retiming constraints over ccf**.** We ensure ρ is a ε-retiming from I_2 to I_1:

$$\begin{aligned} SMT_{\mathrm{retime}_1} = &\forall \chi \in [a, b]\ .\ \forall c_2(\chi) \in I_2\ .\ \exists c_1(\chi) \in I_1\ . \\ &\Big(\rho(c_2(\chi)) = c_1(\chi)\big) \wedge \big(|c_1(\chi) - c_2(\chi)| < \varepsilon\Big) \end{aligned}$$

And that ρ is always increasing:

$$\begin{aligned} SMT_{\mathrm{retime}_2} = &\forall \chi, \chi' \in [a, b]\ .\ \forall c_2(\chi), c_2(\chi') \in I_2\ . \\ &\Big(c_2(\chi) < c_2(\chi') \Rightarrow \rho(c_2(\chi)) < \rho(c_2(\chi'))\Big) \end{aligned}$$

When there are more than 2 agents , we must also encode the constraint that for all $n \neq m$, $\rho_m^{-1} \circ \rho_n$ is an ε-retiming. Thus, for all



(a) $\varphi\ \mathcal{U}_{[a,b]} \psi$ (b) $\varphi\ \mathcal{R}_{[a,b]} \psi$

(c) $\square_{[a,b]}\ \varphi$ (d) $\Diamond_{[a,b]}\ \varphi$ (e) $\neg p$

Figure 5: Conversion of STL syntax trees to their corresponding SMT syntax tree.

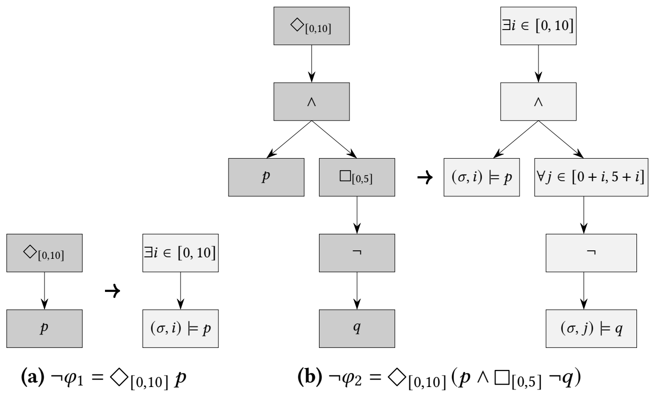
$n \neq m$, denoting f_m as the uninterpreted function that represents the inverse of the uninterpreted c_m:

$$SMT_{\text{retime}_3} = \forall t \in I_n \,.\, f_m(\rho_n(t)) = t$$

Thus, the SMT constraint for signal retiming if the following:

$$\boxed{SMT_{\text{retime}} = SMT_{\text{retime}_1} \;\wedge\; SMT_{\text{retime}_2} \;\wedge\; SMT_{\text{retime}_3}.}$$

Constraints for the STL formula. Let γ_φ be the *syntax tree* representation of an STL formula φ, where each internal node represents an operator, and each leaf node represents an atomic proposition. We convert γ_φ to its *SMT syntax tree* representation τ_φ. An SMT syntax tree τ_φ is a tree obtained from an STL syntax tree γ_φ by replacing each temporal operator in the non-leaf node of γ_φ with its corresponding SMT encoding. In τ_φ as well, each leaf represents an atomic proposition. The purpose of converting an STL formula φ to its SMT syntax tree representation τ_φ is to be able to easily manipulate the syntax tree and parse its corresponding SMT encoding. Figure 5 shows the process of converting all five subtrees with STL operators to their corresponding SMT syntax tree representations. For nested formulas, this process is done for every formula in the STL syntax tree, starting from the root of the tree.



(a) $\neg\varphi_1 = \Diamond_{[0,10]}\, p$ (b) $\neg\varphi_2 = \Diamond_{[0,10]}\,(p \wedge \square_{[0,5]}\,\neg q)$

Figure 6: SMT syntax tree of STL formulas $\neg\varphi_1$ and $\neg\varphi_2$.

For example, Figs. 6a and 6b show creating SMT syntax trees of $\neg\varphi_1$ and $\neg\varphi_2$ using the technique shown in Fig. 5. Let $\tau_{\neg\varphi_1}$ (resp., $\tau_{\neg\varphi_2}$) be the SMT syntax trees created from $\neg\varphi_1$ (resp., $\neg\varphi_2$). Let us first consider the case where the monitor has the whole distributed signal $(E, \leadsto)$ (i.e., no segmentation). The case of a segmented signal will be handled by formula progression explained in Section 5.3. Thus, we keep the SMT syntax trees unchanged and we denote the corresponding SMT constraint by SMT_{τ_φ}. From Fig. 6a, for $\neg\varphi_1$, the distributed signal $(E, \leadsto)$, and the SMT syntax tree $\tau_{\neg\varphi_1}$, we have:

$$SMT_{\tau_{\neg\varphi_1}} = \exists i \in [0,10].((\text{ccf}, i) \models p).$$

Recall from the beginning of this section '$(\text{ccf}, i) \models p$' is replaced with the $f(.) > 0$ in the SMT constraint. For $\neg\varphi_2$, we have:

$$SMT_{\tau_{\neg\varphi_2}} = \exists i \in [0,10].((\text{ccf}, i) \models p \wedge \forall j \in [0{+}i, 5{+}i](\neg((\text{ccf}, j) \models q))).$$

Putting everything together. The final SMT constraint is the following:

$$\boxed{FinalSMT = SMT_{\text{flow}} \;\wedge\; SMT_{\text{retime}} \;\wedge\; SMT_{\tau_{\neg\varphi}}.}$$

Obviously, since there is logical equivalence between an STL formula φ and its corresponding SMT encoding SMT_{τ_φ}, for any given a distributed signal $(E, \leadsto)$ over N agents, we have $(E, \leadsto) \not\models \varphi$ if and only if *FinalSMT* is satisfiable (assuming all time intervals of temporal operators are within $[0, |(E, \leadsto)|]$).

5.3 Formula Progression

We now consider the case where the monitor does not have the entire distributed signal and receives it in segments, or, time intervals of some temporal operators are not within $[0, |(E, \leadsto)|]$. Given a segment $(E, \leadsto)$ and formula φ, our goal is to obtain a *progressed formula* φ' such that any (finite or infinite) extension $(E', \leadsto)$ will be evaluated for φ'.

We define function Λ, that takes as input an SMT syntax tree τ_φ and a segment duration $|(E, \leadsto)|$ and returns as output (see Algorithm 1) an SMT syntax tree $\tau'_\varphi = \Lambda(\tau_\varphi, |(E, \leadsto)|)$. We construct an SMT syntax tree τ'_{φ_μ} from τ'_φ such that the following properties hold:

- The root of τ'_{φ_μ} is the topmost (and leftmost if there are two) node of τ'_φ which has a quantifier label.
- For every subsequent nodes, in τ'_{φ_μ}, if the node n has the label $\wedge$ or $\vee$ with children labelled with quantifiers, remove the node and only keep the left child by doing $n.parent = n.leftchild$.

As examples, let us partition the SMT syntax trees in $\tau_{\neg\varphi_1}$ (Fig. 6a) and $\tau_{\neg\varphi_2}$ (Fig. 6b) at time $t = 5$ using Algorithm 1. For $\tau_{\neg\varphi_1}$, since the starting node n_τ, which is the root node in this case, is labelled '$\exists i \in [0,10]$', we create a node n'_τ and label it '$\vee$' (line 9). Now we create two copies of the tree at n_τ, change the ranges to '$[0,5)$' (resp., '$[5,10]$'), and attach them to left (resp., right) children of n'_τ (lines 10 to 13). n'_τ is our new n_τ (line 17). Now, we repeat the process for each child of n_τ. However, as none of the children nodes are labelled with quantifiers, $\tau'_{\neg\varphi_1} = n_\tau$ is our desired partitioned tree from $\tau_{\neg\varphi_1}$ at time $t = 5$, shown in Fig. 7a. Following the same process, we get $\tau'_{\neg\varphi_2}$ as our partitioned tree from $\tau_{\neg\varphi_2}$ at time $t = 5$, shown in Fig. 7b.

LEMMA 5.1 (SMT PARTITION TREE LEMMA). *Let $(E, \leadsto)$ be a distributed signal and φ be an STL formula. FinalSMT for $(E, \leadsto)$ and τ_φ*

Algorithm 1: Function Λ

```
Data: SMT syntax tree $\tau_\varphi$, partition time $t$
Result: SMT syntax tree $\tau'_\varphi$
Let $root_\tau$ be the root node of $\tau_\varphi$ and $n_\tau$ be a node
Function PartitionTree($n_\tau$):
    if $n_\tau$ has a quantifier with range '$[a, b]$' then
        if $a < t \leq b$ then
            Let $n'_\tau$ be an empty node
            if $n_\tau$ has quantifier '$\forall$' then
                Label $n'_\tau$ as '$\wedge$'
            if $n_\tau$ has quantifier '$\exists$' then
                Label $n'_\tau$ as $\vee$
            $n'_\tau.leftchild \leftarrow$ copy subtree rooted at $n_\tau$
            Set '$[a, \min(b, t))$' as the quantifier range of $n'_\tau.l$
            $n'_\tau.rightchild \leftarrow$ copy subtree rooted at $n_\tau$
            Set '$[\max(a, t), b]$' as the quantifier range of $n'_\tau.r$
            if $n_\tau \neq root_\tau$ then
                $n_\tau.parent.child \leftarrow n'_\tau$
            else
                $n_\tau \leftarrow n'_\tau$
    foreach $n_\tau.child$ in $n_\tau$ do
        PartitionTree($n_\tau.child$)
return PartitionTree($root_\tau$)
```

is satisfiable if and only if FinalSMT for $(E, \rightsquigarrow)$ *and* $\Lambda(\tau_\varphi, |(E, \rightsquigarrow)|)$ *is satifiable.*

Given a distributed signal $(E', \rightsquigarrow)$ and an STL formula φ, the following theorem shows that the subtree τ'_{φ_μ} of $\Lambda(\tau_{\neg\varphi}, |(E, \rightsquigarrow)|)$ allows computing the progressed formula by discharging τ'_{φ_μ}.

Theorem 5.2 (Partial evaluation theorem). *Let* $(E, \rightsquigarrow)$ *be a distributed signal and* φ *be an* STL *formula. It is the case that* $(E, \rightsquigarrow) \models \varphi_\mu$ *if and only if FinalSMT for* $(E, \rightsquigarrow)$ *and* τ'_{φ_μ} *is satisfiable.*

Simply evaluating *FinalSMT* for $(E, \rightsquigarrow)$ and τ'_{φ_μ} is not enough, as we must ensure that there is no loss of information when modifying τ'_φ using the said evaluation results. For example, in Fig. 7b, Since $(\sigma, j_2) \models \neg q$ cannot be evaluated on the first segment, finding only one value of i_1 in this segment may lead to loss of information, as this may ignore other valid values of i_1 that are required to evaluated $(\sigma, j_2) \models \neg q$ on the next segment.

Note that any modification to τ'_φ would naturally occur only in its τ'_{φ_μ} subtree. To this end, we define a function v, that takes as inputs an SMT syntax tree τ'_{φ_μ} and a distributed signal $(E, \rightsquigarrow)$, and returns an SMT syntax tree τ'_{φ_v}, such that, upon replacing τ'_{φ_μ} with τ'_{φ_v} in τ'_φ, τ'_φ can sufficiently evaluate $(E', \rightsquigarrow)$. In other words, the STL representation of τ'_φ becomes the desired progression of φ on $(E, \rightsquigarrow)$. However, before defining v, we specify the following shorthand notations we will be using throughout its definition:

- '$\tau_\varphi = p$': The root of the tree τ_φ is labelled $p \in \mathsf{AP}$.
- $\tau_\varphi = \tau_{\varphi_1} X \tau_{\varphi_2}$, where $X = \{\wedge, \vee\}$: The root of the tree τ_φ is labelled X, and it has two children τ_{φ_1} and τ_{φ_2}.
- $\tau_\varphi = \Box_{[a,b]}\, \tau_\psi$: The root of the tree τ_φ contains label $\forall i \in [a, b]$, and it has a child τ_ψ.
- $\tau_\varphi = \Diamond_{[a,b]}\, \tau_\psi$: The root of the tree τ_φ contains label $\exists i \in [a, b]$, and it has a child τ_ψ.
- $((E, \rightsquigarrow), t) \models \tau_\varphi$: At time instance t, *FinalSMT* for $(E, \rightsquigarrow)$ and τ_φ is satisfiable.

Now we define v in a case-by-case manner for the relevant STL operators:

Atomic propositions. Let $\tau_{\varphi_\mu} = p$ for some $p \in \mathsf{AP}$. We have:

$$v((E, \rightsquigarrow), \tau_{\varphi_\mu}) = \begin{cases} \top & \text{if } ((E, \rightsquigarrow), 0) \models p \\ \bot & \text{otherwise} \end{cases}$$

Conjunction. Let $\tau_{\varphi_\mu} = \tau_{\varphi_{\mu_1}} \wedge \tau_{\varphi_{\mu_2}}$. We have:

$$v((E, \rightsquigarrow), \tau_{\varphi_\mu}) = v((E, \rightsquigarrow), \tau_{\varphi_{\mu_1}}) \wedge v((E, \rightsquigarrow), \tau_{\varphi_{\mu_2}})$$

Disjunction. Let $\tau_{\varphi_\mu} = \tau_{\varphi_{\mu_1}} \vee \tau_{\varphi_{\mu_2}}$. We have:

$$v((E, \rightsquigarrow), \tau_{\varphi_\mu}) = v((E, \rightsquigarrow), \tau_{\varphi_{\mu_1}}) \vee v((E, \rightsquigarrow), \tau_{\varphi_{\mu_2}})$$

Always operator. Let $\tau_{\varphi_\mu} = \Box\, \tau_{\varphi'_\mu}$. In this case, the transformation of τ_{φ_μ} is fairly straightforward:

$$v((E, \rightsquigarrow), \tau_{\varphi_\mu}) = \begin{cases} \Box_{[a,b]}\, \tau_{\varphi'_\mu} & \text{if } \forall k \in [a, b].((E, \rightsquigarrow), k) \models \tau_{\varphi'_\mu} \\ \bot & \text{if } \exists k \in [a, b].((E, \rightsquigarrow), k) \not\models \tau_{\varphi'_\mu} \end{cases}$$

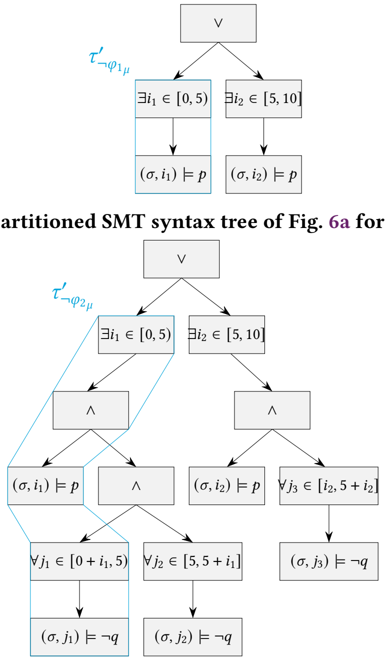
Eventually operator. Let $\tau_{\varphi_\mu} = \Diamond\, \tau_{\varphi'_\mu}$. In this case, instead of finding a single time instance where *FinalSMT* for $(E, \rightsquigarrow)$ and $\tau_{\varphi'_\mu}$ is satisfiable, a valid range $[k, b]$ must be identified, where $k \in [a, b]$ is the earliest time instance where *FinalSMT* for $(E, \rightsquigarrow)$ and $\tau_{\varphi'_\mu}$ is satisfiable:

$$v((E, \rightsquigarrow), \tau_{\varphi_\mu}) = \begin{cases} \Diamond_{[k,b]}\, \tau_{\varphi'_\mu} & \text{if } \operatorname{argmin}_{k \in [a,b]}(((E, \rightsquigarrow), k) \models \tau_{\varphi'_\mu}) \\ \bot & \text{if } \forall k \in [a, b].((E, \rightsquigarrow), k) \not\models \tau_{\varphi'_\mu} \end{cases}$$

Remark 1. *Since Until (Fig. 5a) and Release (Fig. 5b) operators are expressed using existential and global quantifiers in SMT syntax trees, the definition of* v *does not need cases for them.*

Now that we have defined v, we state the necessary steps required to compute the progression of some STL formula φ on a distributed signal $(E, \rightsquigarrow)$ as follows:

- First, we create the SMT syntax tree τ_φ that corresponds to the STL formula φ using the methods detailed in Fig. 5. As examples, let us consider the SMT syntax trees for the STL formulas, $\neg\varphi_1 = \Diamond_{[0,10]}\, p$ (Fig. 6a) and $\neg\varphi_2 = \Diamond_{[0,10]}(p \wedge \Box_{[0,5]}\, \neg q)$ (Fig. 6b).
- Next, we partition τ_φ at time $|(E, \rightsquigarrow)|$ using Algorithm 1, and obtain $\tau'_\varphi = \Lambda(\tau_\varphi, |(E, \rightsquigarrow)|)$, such that τ_{φ_μ} is the subtree in τ_φ that can be evaluated on $(E, \rightsquigarrow)$. In our example, we consider the case where the monitor only has the first 5 time units, that is, $|(E, \rightsquigarrow)| = 5$. Fig. 7a (resp., Fig. 7b) shows the partitioned SMT syntax tree for Fig. 6a (resp., Fig. 6b) at time instance $|(E, \rightsquigarrow)| = 5$ with subtrees $\tau'_{\neg\varphi_{1_\mu}}$ (resp., $\tau'_{\neg\varphi_{2_\mu}}$) that can be evaluated on $(E, \rightsquigarrow)$.
- Finally, we partially evaluate φ on $(E, \rightsquigarrow)$ by transforming τ'_{φ_μ} to $\tau'_{\varphi_v} = v((E, \rightsquigarrow), \tau'_{\varphi_\mu})$. The STL representation of this new SMT syntax tree τ'_φ is our desired progression of φ on the extension of $(E, \rightsquigarrow)$. In our first example, $\neg\varphi_{1_p}$ is of the form $\Diamond_{[0,5]}\, \neg\varphi'_{1_p}$. Now, let us assume that p is never `true` in $(E, \rightsquigarrow)$. In that case, according to the rules specified for v, The label of the root of $\tau'_{\neg\varphi_{1_\mu}}$ stays unchanged, and the child becomes `false`. Therefore, the progression becomes $(\Diamond_{[0,5]}\, \texttt{false}) \vee (\Diamond_{[5,10]}\, p)$, which is, $\Diamond_{[5,10]}\, p$ upon simplification. In our second example, $\neg\varphi_{2_p}$ is of the form $\Diamond_{[0,5]}\, \neg\varphi'_{2_p}$. Now, let us assume that the minimum i


$\tau'_{\neg\varphi_{1_\mu}}$
∨
$\exists i_1 \in [0,5)$
$\exists i_2 \in [5,10]$
$(\sigma, i_1) \models p$
$(\sigma, i_2) \models p$
$\tau'_{\neg\varphi_{2_\mu}}$
∨
$\exists i_1 \in [0,5)$
$\exists i_2 \in [5,10]$
∧
∧
$(\sigma, i_1) \models p$
∧
$(\sigma, i_2) \models p$
$\forall j_3 \in [i_2, 5+i_2]$
$\forall j_1 \in [0+i_1, 5)$
$\forall j_2 \in [5, 5+i_1]$
$(\sigma, j_3) \models \neg q$
$(\sigma, j_1) \models \neg q$
$(\sigma, j_2) \models \neg q$

(a) Partitioned SMT syntax tree of Fig. 6a for $\tau'_{\neg\varphi_1}$.

(b) Partitioned SMT syntax tree of Fig 6b for $\tau'_{\neg\varphi_2}$.

Figure 7: Examples of partitioned SMT syntax tree of STL formulas $\neg\varphi_1$ and $\neg\varphi_2$ at $t = 5$.

for which $\exists i \in [0,5)((((E, \rightsquigarrow), i) \models p) \wedge (\forall j \in [i+0, \min(i+5,5)]((E, \rightsquigarrow), i) \models \neg q)))$ is satisfied at time 3.5. In that case, according to the rules specified for υ, The label of the root of $\tau'_{\neg\varphi_{2_\mu}}$ is changed to $\exists i_1 \in [3.5, 5)$. Therefore, the progression becomes $(\Diamond_{[3.5,5)}(\Box_{[0,5]} \neg q)) \vee (\Diamond_{[5,10]}(p \wedge (\Box_{[0,5]} \neg q)))$.

6 CASE STUDIES AND EVALUATION

In this section, we evaluate our algorithm for monitoring STL specifications on distributed signals using two case studies. The source codes related to our experiments can be found at: https://github.com/A-N-I-K/CPS_STL_Prog_RE_Package.

6.1 Case Studies

6.1.1 Network of UAVs. We use the Fly-by-Logic framework [23], a path planner software for UAVs, to simulate flight paths of two UAVs that take off after 1.5s, hover, and then land after 4.5s. The trajectories are sampled at 20Hz as x_n, y_n, and z_n coordinates for each UAV with an ε ranging between 1 to 5ms.

6.1.2 Water Distribution System. We consider a hybrid water distribution system consisting of two tanks as shown in Figure 1. Each tank has an inlet pipe connected to an external water source, and an outlet pipe with a valve used to regulate high pressure water outflow. A controller on each tank operates its valve, and samples the outflow pressure at 20Hz using its local clock. We model such a system in Simulink to emulate the Refueling Water Storage Tanks (RWST) module of an Emergency Core Cooling System (ECCS) of a Pressurized Water Reactor Plant [28]. ECCS provides core cooling to minimize fuel damage following 'loss of coolant' incidents by administering high pressure water injection from RWST. The tanks and their controllers operate even when the supply of power is lost to the plant. As a failsafe, ECCS incorporates Cold Leg Accumulators (CLA) that do not require power to operate. These tanks contain large amounts of borated water with a pressurized nitrogen gas bubble at the top. If the outflow pressure drops below a certain threshold, the nitrogen forces borated water out of the tank and into the reactor coolant system. A reasonable range for ε here is 5 to 500ms [3].

6.2 Experimental Setup

In our UAV related experiments, we monitor three STL properties: (1) mutual separation between UAVs never falls below a threshold; (2) all UAVs take off simultaneously from standby state and hover at the same altitude, and (3) all UAVs eventually land simultaneously. The monitor receives a distributed signal every second, and we measure its execution time for each formula progression to verify truthfulness of the given formulas. In our water tank related experiments, we simulate a plant failure where the RWST in the ECCS is triggered upon receiving an emergency actuation signal. The monitor receives a distributed signal at varying time intervals from multiple water tanks. Our goal is to find possible violations caused by clock drift, where the water pressure falls below threshold required to keep the failsafe CLA from triggering. All experiments are replicated to exhibit 95% confidence interval to provide statistical significance. The experimental platform is a CentOS server with an Intel(R) Xeon(R) Platinum 8180 CPU @ 3.80GHz clock rate and 754G of RAM. Our implementation invokes the SMT-solver Z3 [21] to solve the problem described in Sections 4 and 5.

6.3 Analysis of Results

Mutual separation. This property states that the distance between every pair of UAVs in fleet always remain above a given threshold δ. The corresponding STL formula φ_{ms} is:

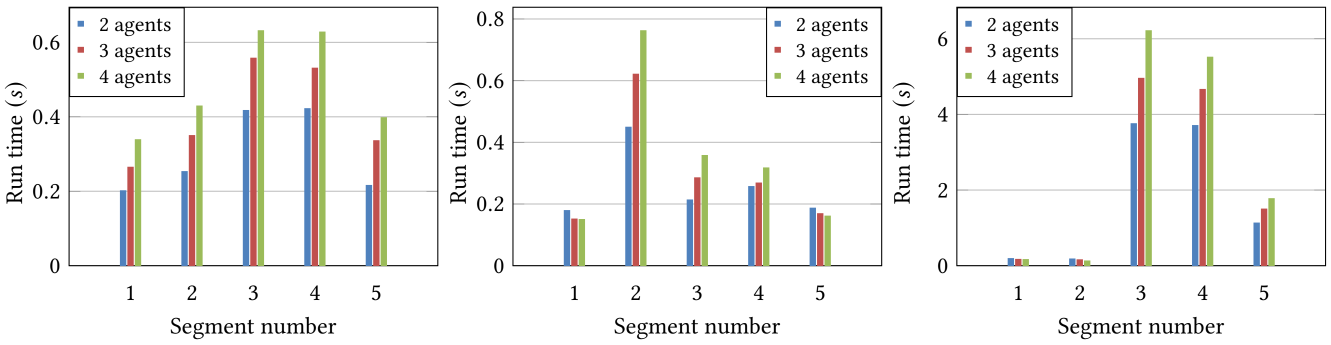
$$\bigwedge_{i,j \in [N], i \neq j} \Box_{[0,\infty]} \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} > \delta\right).$$

Figure 8a shows the run time for each segment for evaluation of φ_{ms} on the distributed signal. In each segment the progression formula remains unchanged. However, the first segment shows minimal run time due to the fact that the UAVs are stationary throughout the entirety of that segment and, therefore, require very few 'unique' distance calculations. The run time for the second segment and the last segment are slightly higher than that of the first segment because of the same reason; the UAVs are partially grounded throughout these two segments. Note that despite φ_{ms} seemingly being a simple STL formula, the average run time per segment is relatively higher (compared to the run time of other formulas) due to requiring quadratic equations to be solved.

Eventually hover. This property states that the UAVs in fleet are eventually (within 2s) airborne and hover within a λ height margin. Formally, the corresponding STL formula φ_{eh} is:

$$\bigwedge_{i,j \in [N], i \neq j} \Diamond_{[0,2]} \left(z_i, z_j > 0\right) \Rightarrow \Box_{[0,\infty]} \left(|z_i - z_j| < \lambda\right).$$

Fig. 8b shows the run time for each segment for evaluation of φ_{eh} on the distributed signal. The first segment has the lowest run time



(a) φ_{ms} **(Mutual separation).** **(b)** φ_{eh} **(Eventually hover).** **(c)** φ_{el} **(Eventually land).**

Figure 8: Effect of segment number and number of agents on run time for different flight properties.

Clock Skew (s)	True Violations	Detected Violations	False Positives	False +ve Percentage
0.05	9	25	16	64%
0.1	4	42	38	90.48%
0.15	12	65	53	81.54%
0.2	11	80	69	86.25%
0.25	4	86	82	95.35%
0.3	7	99	92	92.93%
0.35	5	112	107	95.54%
0.4	7	127	120	94.49%
0.45	10	145	135	93.1%
0.5	7	160	153	95.63%

(a) Water tanks.

Clock Skew (s)	True Violations	Detected Violations	False Positives	False +ve Percentage
0.05	6	11	5	45.45%
0.1	6	20	14	70%
0.15	8	30	22	73.33%
0.2	4	39	35	89.74%
0.25	2	46	44	95.65%
0.3	1	48	47	97.92%
0.35	7	62	55	88.71%
0.4	2	66	64	96.97%
0.45	5	76	71	93.42%
0.5	6	84	78	92.86%

(b) UAVs.

Table 1: Impact of ε.

as the UAVs are stationary. The second segment has a higher run time because $(z_i, z_j > 0)$ is observed and progression is needed for the following segments, where the progressed formula simply becomes $\Box_{[0,\infty]}(z_i = z_j)$.

Eventually land This property states that the UAVs in fleet eventually land on the ground simultaneously. Formally, the corresponding STL formula φ_{el} is:

$$\bigwedge_{i,j\in[N],i\neq j} \Diamond_{[2,\infty]}\Big(z_i = 0 \wedge z_j = 0\Big).$$

Fig. 8c shows the run time for each segment for evaluation of φ_{el} on the distributed signal. The temporal interval of φ_{el} is intentionally $[2,\infty]$ instead of $[0,\infty]$ since the UAVs are on the ground at the start of the distributed signal. The behavior in run time shown in this figure is opposite of what we have witnessed in Fig. 8b. In segments 3 and 4, the UAVs are airborne, and therefore, the search-space for the SMT problem is exhaustively traversed. However, in segment 5, φ_{el} is satisfied and the progression becomes `true`.

Impact of segment duration and number of water tanks. Let $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_N$ denote the outflow pressures of N number of water tanks. For simplicity, we assume all the pipes are of the same diameter. Thus, the pressure exerted on the CLA is $\mathbf{P}_1 + \mathbf{P}_2 + \ldots + \mathbf{P}_N$. We monitor the property that states outflow pressure remains above the threshold pressure 600*psig* [27] indefinitely. The corresponding STL formula $\varphi_{\mathbf{P}}$ is:

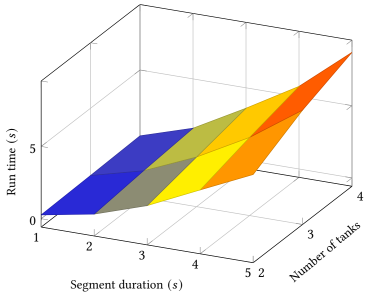


Figure 9: Effect of segment duration and number of water tanks on run time for $\varphi_{\mathbf{P}}$.

$$\Box_{[0,\infty]}\Bigg(\sum_{n=1}^{N} P_n \geq 600\Bigg).$$

Fig. 9 shows the effect on run time for increasing the number of tanks from 2 to 4 with $\varepsilon = 0.05s$ over segment duration ranging from 1s to 5s. As expected, both segment duration and the number of tanks drive up the run time. We note that even when the monitor receives the distributed signals sent by the water tanks at a reasonable 1s intervals, the monitor is still able to verify the property online under around half a second for four tanks.

Impact of clock skew. In order to study the impact of ε on monitoring verdicts, we model two RWST modules with intentional 'faults', where the outflow pressures of either tank can drop below the threshold pressure of the CLA. Thus, if both tanks' pressures fall simultaneously, the CLA gets triggered. We also introduce a clock drift in the valve controller of one of the tanks. Table 1a shows the results for two tanks that were active for an hour. During this time, Tank 1 and Tank 2 reported low pressures for a total of 35.5s and 36.1s respectively. Although generally we are interested in finding a single violation, in order to demonstrate the effect of clock skew, we find multiple violation instances in this experiment by tallying up pairs of piece-wise linear interpolations between samples where violations are detected. We report the number of *true violations* as a baseline that was reverse calculated from the introduced clock drift ε, number of *detected violations* using our method, and the number of *false positives*, which is essentially the difference between the true violations and the detected violations[1]. Note that there are no *false negatives*. Furthermore, as the clock drift is increased from 0.05s to 0.5s, the number of false positives increase as well. Similarly, we model a path for a pair of UAVs, where the agents periodically reside within the given mutual separation threshold, and violate the mutual separation property. Table 1b shows the results for two UAVs in operation for half an hour. We again report the number of true violations, detected violations, and false positives.

7 CONCLUSION

In this paper, we introduced a technique for monitoring specifications expressed in the STL for distributed CPS, where continuous-time and valued signals from a set of agents do not share a global clock. Our technique assumes an off-the-shelf clock synchronization algorithm (such as NTP) that ensures a maximum bounded clock skew among all the agents in the system. We also introduced a signal *retiming* technique that efficiently aligns continuous signals to detect possible violations of STL specifications. We reduce our runtime monitoring problem to an *SMT solving problem* and introduce a *formula progression* technique that takes a distributed signal and an STL formula as input and returns another STL formula as output that represents the progression of the formula over the signals. We also reported experimental results on monitoring a fleet UAVs, as well as a water distribution system.

For future research, mapping fragments of STL to the right class of complexity is a natural next step, as there may be cases where the complexity of monitoring depend on the complexity hierarchy of SMT solving. Incorporating a fully distributed monitoring framework could be another possible focus. Furthermore, as monitors in the system may be subject to faults, developing distributed *fault-tolerant* monitoring algorithms could be another research avenue.

ACKNOWLEDGMENTS

This work is sponsored by the United States National Science Foundation (NSF) awards CCF (SHF) 2118356 and CCF (FMitF) 2102106.

REFERENCES

[1] Abbas, H., Mittelmann, H., Fainekos, G.: Formal property verification in a conformance testing framework. In: ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE) (October 2014)

[2] Bauer, A., Falcone, Y.: Decentralised LTL monitoring. Formal Methods in System Design **48**(1-2), 46–93 (2016)

[3] Benndorf, M., Haenselmann, T.: Time synchronization on android devices for mobile construction assessment. In: The Tenth International Conference on Sensor Technologies and Applications. Thinkmind (2016)

[4] Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems **3**(1), 63–75 (1985)

[5] Chauhan, H., Garg, V.K., Natarajan, A., Mittal, N.: A distributed abstraction algorithm for online predicate detection. In: Proceedings of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS). pp. 101–110 (2013)

[6] Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. Formal Methods in System Design **49**(1-2), 109–158 (2016)

[7] Danielsson, L.M., Sánchez, C.: Decentralized stream runtime verification. In: Proceedings of the 19th International Conference on Runtime Verification (RV). pp. 185–201 (2019)

[8] Dokhanchi, A., Hoxha, B., Fainekos, G.: Online monitoring for temporal logic robustness. In: Proc. of Runtime Verification (2014)

[9] Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Proceedings of the 25th International Conference on Computer Aided Verification (CAV). pp. 264–279 (2013)

[10] Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS). pp. 92–106 (2010)

[11] Fainekos, G.E., Pappas, G.J.: Robust sampling for MITL specifications. In: Proceedings of 5th International Conference on the Formal Modeling and Analysis of Timed Systems (FORMATS). pp. 147–162 (2007)

[12] Ganguly, R., Momtaz, A., Bonakdarpour, B.: Distributed runtime verification under partial asynchrony. In: Proceedings of the 24nd International Conference on Principles of Distributed Systems (OPODIS). pp. 20:1–20:17 (2020)

[13] J, V.D., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods System Design **51**(1), 5–30 (2017)

[14] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(7), 558–565 (1978)

[15] Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proceedings of the Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS/FTRTFT). pp. 152–166 (2004)

[16] Medhat, R., Bonakdarpour, B., Kumar, D., Fischmeister, S.: Runtime monitoring of cyber-physical systems under timing and memory constraints. ACM Transactions of Embedded Computing Systems **14**(4), 79:1–79:29 (2015)

[17] Mills, D.: Network time protocol version 4: Protocol and algorithms specification. RFC 5905, RFC Editor (June 2010)

[18] Mittal, N., Garg, V.K.: Techniques and applications of computation slicing. Distributed Computing **17**(3), 251–277 (2005)

[19] Momtaz, A., Basnet, N., Abbas, H., Bonakdarpour, B.: Predicate monitoring in distributed cyber-physical systems. In: Proceedings of the 21st International Conference on Runtime Verification (RV) 2021. pp. 3–22 (2021)

[20] Mostafa, M., Bonakdarpour, B.: Decentralized runtime verification of LTL specifications in distributed systems. In: Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 494–503 (2015)

[21] de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 337–340 (2008)

[22] Ogale, V.A., Garg, V.K.: Detecting temporal logic predicates on distributed computations. In: Proceedings of the 21st International Symposium on Distributed Computing (DISC). pp. 420–434 (2007)

[23] Pant, Y.V., Abbas, H., Mangharam, R.: Smooth operator: Control using the smooth robustness of temporal logic. In: IEEE Conference on Control Technology and Applications, (2017)

[24] Pereira, Carlos, J., Machado, N., Pinto, J.S.: Testing for race conditions in distributed systems via smt solving. In: International Conference on Tests and Proofs, Bergen, Norway, June 22-26, 2020, Proceedings. pp. 122–140 (2020)

[25] Quesel, J.D.: Similarity, Logic, and Games: Bridging Modeling Layers of Hybrid Systems. Ph.D. thesis, Carl Von Ossietzky Universitat Oldenburg (July 2013), http://www.cs.cmu.edu/~jquesel/paper/diss.pdf

[26] Sen, K., Vardhan, A., Agha, G., G.Rosu: Efficient decentralized monitoring of safety in distributed systems. In: ICSE (2004)

[27] USNRC: Emergency core cooling systems (March 2021), https://www.nrc.gov/docs/ML1122/ML11223A220.pdf

[28] USNRC: Pressurized water reactor systems (March 2021), https://www.nrc.gov/reading-rm/basic-ref/students/for-educators/04.pdf

[29] Valapil, V.T., Yingchareonthawornchai, S., Kulkarni, S.S., Torng, E., Demirbas, M.: Monitoring partially synchronous distributed systems using SMT solvers. In: Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings. pp. 277–293 (2017)

A PROOFS

We include the proofs for all the theorems and lemmas in our paper below. In order to keep the text clear and concise, we shorten the notation of a distributed signal from $(E, \rightsquigarrow)$ to E from here on out.

[1] We emphasize that due to the uncertainty caused by asynchrony, the existence of false positives is inevitable, as there is no global clock to ensure a total order of events.

A.1 Lemma 4.1

Proof. Assume $|\min I_n - \min I_m| > \varepsilon$. However, both $\min I_n$ and $\min I_m$ are lower bounds of I_n and I_m respectively, at the k^{th} transmission. Therefore, by definition of partial synchrony, the difference of their values mustnot exceed the maximum closk skew ε. Therefore, our assumption is not possible. Thus, $|\min I_n - \min I_m| \leq \varepsilon$. Similarly, we can show that $|\max I_n - \max I_m| \leq \varepsilon$. □

A.2 Theorem 4.4

Proof. We distinguish two cases:

- (⇐) Suppose that such retimings exist. We define local time values for each time $\chi \in [t+a, t+b]$ for agents $A_1, A_2, \ldots, A_N$ respectively as $t_1^{\chi} = c_1(\chi)$ and $t_n^{\chi} = \rho_n^{-1}(c_1(\chi))$, where $2 \leq n \leq N$. In other words, t_n^{χ} is the local time of agent A_n at global time χ. Furthermore, define $C_\chi = \{(t_n, x_n(t_n)) \mid t_n \leq t_n^{\chi} \land n \in [N]\}$. By the construction of C_χ, and the fact that the retimings respect $\rightsquigarrow$, it holds that if $e \in C_\chi$ and $f \rightsquigarrow e$, then $f \in C_\chi$. For every $n, m \geq 2$ and $n \neq m$, it holds that $t_m^{\chi} = \rho_m^{-1}(\rho_n(t_n^{\chi}))$ so $|t_n^{\chi} - t_m^{\chi}| \leq \varepsilon$. Thus, C_χ is a consistent cut, and the flow of frontiers $\mathrm{front}(C_\chi)$, where $\chi \in \mathbb{R}_+$, is a consistent cut flow $\sigma \in \mathcal{CCF}$ that witnesses the violation of φ.
- (⇒) Suppose $\sigma \in \mathcal{CCF}$ is a consistent cut flow that violates φ. By definition, there must be a sequence of consistent cuts in σ that violates φ. Let C_χ denote the last cut in the flow (which by definition contains all the cuts in the flow) and let $\mathrm{front}(C_\chi)$ denote the frontier of all these consistent cuts in the flow. For every two events $(t_n, x_n(t_n))$ and $(t_m, x_m(t_m))$ in $\mathrm{front}(C_\chi)$, we have $|t_n - t_m| \leq \varepsilon$. Since $(t_n, x_n(t_n)) \in \mathrm{front}(C_\chi)$, we have $(s, x_m(s)) \in C_\chi$ for all s s.t. $s + \varepsilon \leq t_n$. Thus, $t_m \geq s$ for all such s and so $t_m \geq t_n - \varepsilon$. By symmetry of the argument, $t_n \geq t_m - \varepsilon$ holds as well, implying a retiming indeed does exist.

□

A.3 Lemma 5.1

Proof. We distinguish the following cases:

Case 1: First, we consider the base case of this proof, where the formula is an atomic proposition, that is, $\varphi = p$.

(⇒) The SMT encoding generated by
for E and τ_p is:

$$(\mathrm{ccf}, 0) \models p$$

In other words, when the encoding above is satisfied, the events in the frontier of the consistent cut at time 0 satisfies p. Now, as the SMT syntax tree for p does not have any quantifiers, Algorithm 1 never enters Line 4. Hence, the SMT syntax tree for p remains unchanged, and the SMT encoding using E and $\tau'_{\varphi} = \Lambda(\tau_\varphi, |E|)$ is:

$$(\mathrm{ccf}, 0) \models p$$

(⇐) Trivial.

Case 2: Assume that the proof has been established for the cases when the formulas are $\varphi = \varphi_1$ and $\varphi = \varphi_2$. Now, we consider the case where the formula is $\varphi = \varphi_1 \land \varphi_2$.

(⇒) The SMT encoding generated by using E and $\tau_{\varphi_1 \land \varphi_2}$ is:

$$(\mathrm{ccf}, 0) \models \varphi_1 \land \varphi_2$$

In other words, when the encoding above is satisfied, the events in the frontier of the consistent cut at time 0 satisfies $\varphi_1 \land \varphi_2$. Now, as the SMT syntax tree for φ does not have any quantifiers, Algorithm 1 never enters Line 4. Hence, the SMT syntax tree for φ remains unchanged, and the SMT encoding using E and $\tau'_{\varphi_1 \land \varphi_2} = \Lambda(\tau_{\varphi_1 \land \varphi_2}, t')$ is:

$$(\mathrm{ccf}, 0) \models (\varphi_1 \land \varphi_2) \land \mathtt{true}$$

(⇐) Trivial.

Case 3: Assume that the proof has been established for the cases when the formulas are $\varphi = \varphi_1$ and $\varphi = \varphi_2$. Now, we consider the case where the formula is $\varphi = \varphi_1 \lor \varphi_2$.

(⇒) The SMT encoding using E and $\tau_{\varphi_1 \lor \varphi_2}$ is:

$$(\mathrm{ccf}, 0) \models \varphi_1 \lor \varphi_2$$

In other words, when the encoding above is satisfied, the events in the frontier of the consistent cut at time 0 satisfies $\varphi_1 \lor \varphi_2$. Now, as the SMT syntax tree for φ does not have any quantifiers, Algorithm 1 never enters Line 4. Hence, the SMT syntax tree for φ remains unchanged, and the SMT encoding using E and $\tau'_{\varphi_1 \lor \varphi_2} = \Lambda(\tau_{\varphi_1 \lor \varphi_2}, t')$ is:

$$(\mathrm{ccf}, 0) \models \varphi_1 \lor \varphi_2$$

(⇐) Trivial.

Case 4: Assume that the proof has been established for the cases when the formulas are $\varphi = \varphi_1$ and $\varphi = \varphi_2$. We consider the case where the formula is $\varphi = \varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2$.

(⇒) The SMT encoding generated by using E and $\tau_{\varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2}$ is:

$$\exists i \in [a,b]\Big((\mathrm{ccf}, i) \models \varphi_2 \land \forall j \in [0,i)(\mathrm{ccf}, j) \models \varphi_1)\Big)$$

If the above encoding is SAT, then both $\exists i \in [a,b]\big((\mathrm{ccf}, i) \models \varphi_2\big)$ and $\exists i \in [a,b] \forall j \in [0,i)\big((\mathrm{ccf}, j) \models \varphi_1\big)$ are SAT. For $a < |E| \leq b$, this can be written as:

$$\exists i_1 \in [a, |E|)\Big((\mathrm{ccf}, i_1) = \varphi_2 \land \big(\forall j_1 \in [0, i_1]((\mathrm{ccf}, j_1) = \varphi_1)\big)\Big)$$

$$\lor$$

$$\exists i_2 \in [|E|, b]\Big((\mathrm{ccf}, i_2) = \varphi_2 \land \big(\forall j_2 \in [|E|, b]((\mathrm{ccf}, j_2) = \varphi_1)\big)\Big)$$

Note that this is the SMT encoding generated by using E and $\tau'_{\varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2} = \Lambda(\tau_{\varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2}, |E|)$, when $a < |E| \leq b$. For any other value of $a < |E| \leq b$, the SMT syntax tree remains unchanged. When the SMT encoding of $\tau_{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}$ is SAT, either (1) $\varphi_1 \mathcal{U}_{[a,|E|]} \varphi_2$ is satisfied, or (2) φ_1 is satisfied throughout $[0, |E|)$, and $\varphi_1 \mathcal{U}_{[|E|,b]} \varphi_2$ is satisfied. If $\varphi_1 \mathcal{U}_{[a,|E|]} \varphi_2$ is satisfied, then the first part of the SMT encoding of $\tau'_{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}$ becomes SAT, and if φ_1 is satisfied throughout $[0, |E|)$, and $\varphi_1 \mathcal{U}_{[|E|,b]} \varphi_2$ is satisfied, then the second part of the SMT encoding of $\tau'_{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}$ becomes SAT. Therefore, in all possible cases, if the SMT encoding of $\tau_{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}$ yields SAT, then the SMT encoding of $\tau'_{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}$ will also yield SAT.

$\Leftarrow$ Trivial.

Case 5: Assume that the proof has been established for the cases when the formulas are $\varphi = \varphi_1$ and $\varphi = \varphi_2$. Finally, we consider the case where the formula is $\varphi = \varphi_1 \mathcal{R}_{[a,b]} \varphi_2$.

($\Rightarrow$) The SMT encoding generated by using E and $\tau_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ is:

$$\exists i \in [a,b] \Big((\text{ccf}, i) \models \varphi_1 \land \forall j \in [0,i) (\text{ccf}, j) \models \varphi_2) \Big)$$

If the above encoding is SAT, then both $\exists i \in [a,b]\big((\text{ccf}, i) \models \varphi_1\big)$ and $\exists i \in [a,b] \forall j \in [0,i) \big((\text{ccf}, j) \models \varphi_2\big)$ are SAT. For $a < |E| \leq b$, this can be written as:

$$\exists i_1 \in [a, |E|) \Big((\text{ccf}, i_1) = \varphi_1 \land \big(\forall j_1 \in [0, i_1] ((\text{ccf}, j_1) = \varphi_2)\big) \Big)$$

$$\lor$$

$$\exists i_2 \in [|E|, b] \Big((\text{ccf}, i_2) = \varphi_1 \land \big(\forall j_2 \in [|E|, b] ((\text{ccf}, j_2) = \varphi_2)\big) \Big)$$

Note that this is the SMT encoding generated by using E and $\tau'_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2} = \Lambda(\tau_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}, |E|)$, when $a < |E| \leq b$. For any other value of $a < |E| \leq b$, the SMT syntax tree remains unchanged. When the SMT encoding of $\tau_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ is SAT, either (1) $\varphi_1 \mathcal{R}_{[a,|E|]} \varphi_2$ is satisfied, or (2) φ_2 is satisfied throughout $[0, |E|)$, and $\varphi_1 \mathcal{R}_{[|E|,b]} \varphi_2$ is satisfied. If $\varphi_1 \mathcal{R}_{[a,|E|]} \varphi_2$ is satisfied, then the first part of the SMT encoding of $\tau'_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ becomes SAT, and if φ_2 is satisfied throughout $[0, |E|)$, and $\varphi_1 \mathcal{R}_{[|E|,b]} \varphi_2$ is satisfied, then the second part of the SMT encoding of $\tau'_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ becomes SAT. Therefore, in all possible cases, if the SMT encoding of $\tau_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ yields SAT, then the SMT encoding of $\tau'_{\varphi_1 \mathcal{R}_{[a,b]} \varphi_2}$ will also yield SAT.

$\Leftarrow$ Trivial.

□

A.4 Theorem 5.2

Proof. Let us assume that $\tau'_{\varphi} = \Lambda(\tau_{\varphi}, |E|)$, $E \models \varphi_\mu$, and *FinalSMT* for $(E, \rightsquigarrow)$ and τ'_{φ_μ} is not satisfiable. This implies that τ'_{φ_μ} has at least one subtree, where the root node is the n^{th} nested quantifier with an interval $[\alpha_n, \beta_n]$ and $\beta_n > |E|$. However, while constructing τ'_{φ_μ}, only the left child is kept for any node that has the label $\land$ or $\lor$ with children labelled with quantifiers (see Section 5). Furthermore, In Line 10 of Algorithm 1, the maximum range of the quantifier labelled on the left child is $\min(\beta_n, |E|)$. Therefore, $\beta_n > |E|$ is not possible. Therefore, such a subtree cannot exist, and by extension τ'_{φ_μ} cannot exist. Thus, $E \models \varphi_\mu$ if and only if *FinalSMT* for $(E, \rightsquigarrow)$ and τ'_{φ_μ} is satisfiable. □