







CauMon: An Informative Online Monitor for Signal Temporal Logic

Zhenya Zhang¹(✉) , Jie An^{2,3}(✉) , Paolo Arcaini³(✉) ,
and Ichiro Hasuo³(✉) 



¹ Kyushu University, Fukuoka, Japan
zhang@ait.kyushu-u.ac.jp

² Institute of Software, Chinese Academy of Sciences,
Beijing, China

anjie@iscas.ac.cn

³ National Institute of Informatics, Tokyo, Japan
{arcaini,hasuo}@nii.ac.jp



Abstract. In this paper, we present a tool for monitoring the traces of cyber-physical systems (CPS) at runtime, with respect to *Signal Temporal Logic* (STL) specifications. Our tool is based on the recent advances of *causation monitoring*, which reports not only whether an executing trace violates the specification, but also *how relevant* the increment of the trace at each instant is to the specification violation. In this way, it can deliver more information about system evolution than classic online robust monitors. Moreover, by adapting two dynamic programming strategies, our implementation significantly improves the efficiency of causation monitoring, allowing its deployment in practice. The tool is implemented as a C++ executable and can be easily adapted to monitor CPS in different formalisms. We evaluate the efficiency of the proposed monitoring tool, and demonstrate its superiority over existing robust monitors in terms of the information it can deliver about system evolution.

Keywords: online monitoring · Signal Temporal Logic · dynamic programming

1 Introduction

Cyber-physical systems (CPS), that embed cyber technologies into physical systems, have been widely deployed in safety-critical domains, such as transportation, healthcare, power and energy. Due to their safety-critical nature, the behaviors of CPS require formal verification to guarantee their satisfaction to formal specifications that are usually expressed in temporal logics, e.g., *Signal Temporal Logic* (STL) [20]. Given an STL specification, *monitoring* (a.k.a. *runtime verification*) [2] is an effective approach for checking whether a trace of system execution satisfies the specification.

Monitoring can be achieved either *offline* or *online*. In STL monitoring, an *offline* monitor can report a real value (called *robustness*) that indicates *how*

robustly an STL formula φ is satisfied or violated by a complete execution trace, based on the STL robust semantics [9, 13]. By contrast, an *online* monitor targets partial execution traces at runtime, reporting the satisfaction of an STL formula φ by the partial trace so far at each time instant. Due to the lack of a complete trace, a typical *online monitor*, e.g., the *robust monitor* in [6], reports a robustness interval $[[R]^L, [R]^U]$ telling the possibly reachable values of the robustness under any suffix trace, where $[R]^L$ and $[R]^U$ are the lower and upper bounds respectively. In this way, the satisfaction of φ can be inferred from the computed robustness interval, e.g., φ is violated if $[R]^U$ is negative.

Online robust monitoring [6] suffers from the *information masking* problem [24, 26, 27], as visualized by the example in Fig. 1. The specification in this example requires that, during $[0, 45]$, the speed v of a car should never be over 10 for 5 time units. The top plot reports a trace v that violates the system specification. When using the classic online monitor [6] reported in the middle plot, the value of $[R]^U$ keeps on decreasing and becomes negative at $b = 10$. It then reaches the minimum value around $b = 14$ and stagnates at that value in the remaining of the monitoring. As a result, the system evolution is not faithfully reflected by the monitor. For instance, the monitor does not deliver that the system actually recovers (i.e., $v < 10$) at $b = 20$ and that the status $v > 10$ persisting for more than 5 time units happens once again from $b = 25$.

Causation monitoring [26] emerges to tackle the information masking issue of robust monitoring. As shown in the bottom plot of Fig. 1, at each instant b , an online causation monitor returns $[\mathcal{R}]^\ominus$ (called a *violation causation distance*) and $[\mathcal{R}]^\oplus$ (called a *satisfaction causation distance*), that respectively reflect how far the trace value at b is from being a *causation* to the violation and the satisfaction of the specification. For instance, $[\mathcal{R}]^\ominus$ at $b = 12$ is negative, which implies that the trace value at $b = 12$ is considered as a *causation* to the violation of the partial trace, because the status $v > 10$ that has been persisting for more than 5 time units continues at $b = 12$. At $b = 20$, $[\mathcal{R}]^\ominus$ becomes positive, which implies that the trace value at $b = 20$ is no more a *causation* to the violation, because v becomes less than 10 at $b = 20$. From Fig. 1, we can see that compared to robust monitoring, causation monitoring can reflect more information about system evolution, such as the recovery of the system at $b = 20$ and the recurrence of the status $v > 10$ persisting for more than 5 time units from $b = 25$.

Contributions. In this paper, we present a tool *CauMon*, that implements an efficient online causation monitoring algorithm of STL. Compared to the plain monitoring algorithm [26] derived directly from the definition of causation monitoring, our algorithm features the use of two dynamic programming strategies,

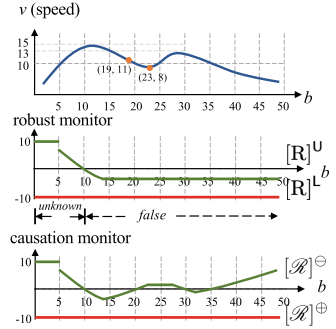


Fig. 1. An illustrative example of online robust monitoring and causation monitoring of STL formula $\varphi : \square_{[0,45]}(\diamond_{[0,5]}v < 10)$.

by which we can significantly reduce redundant computation of the causation distances during monitoring of system executions. We adopt dynamic programming for two purposes: first, we record and reuse the intermediate monitoring results of the sub-formulas using several worklists for each of the sub-formulas, such that computational cost is only spent for incremental results, not for existing ones; moreover, we adapt a sliding window algorithm [18] to accelerate the computation of monitoring results in the presence of nested temporal operators.

We implement **CauMon** in **C++**, and it can be compiled to an executable that can be easily interfaced with CPS in different formalisms. We demonstrate the advantages of **CauMon** in informativeness, by comparing it with the existing online robust monitor in [6]. Moreover, we also evaluate the efficiency of **CauMon**, by comparing it with the plain causation monitor derived from the definition of causation monitoring directly, and also the online robust monitor [6]. The experimental results show that **CauMon** can indeed deliver more information about system evolution compared to the robust monitor; moreover, while the plain implementation of causation monitoring is not applicable to handling nested temporal operators in practice, **CauMon** can significantly reduce the monitoring time costs and achieve comparable efficiency with the robust monitor.

Related Work. Online monitoring is an approach that monitors system executions at runtime, and different approaches have been proposed for different temporal logics, such as LTL [4, 5], MTL [14, 19, 25], and STL [6, 7, 15, 16, 23]. For online monitoring of STL, most of existing approaches [6, 7, 15, 16] and tools [1, 8, 22, 23] are based on its robust semantics and provide a quantitative value or interval to characterize the system runtime status. Consequently, these approaches suffer more or less from the issue of information masking. In [27], we proposed a reset mechanism that resets a monitor whenever it detects the recovery of specification violation. However, [27] does not propose new semantics and so it does not improve informativeness of monitors between two resets.

2 Preliminaries

2.1 Signal Temporal Logic

Let $T \in \mathbb{R}_+$ be a positive real. A *signal* (i.e., a trace of system execution) is a function $\mathbf{v}: [0, T] \rightarrow \mathbb{R}^d$, where T is the *time horizon*, $d \in \mathbb{N}_+$ is the dimension. In practice, each signal dimension concerns with a *signal variable* that has a certain physical meaning, e.g., **speed**, **RPM** of a car. We fix a set **Var** of variables and assume that a signal \mathbf{v} is *spatially bounded* by a hyper-rectangle Ω , i.e., for any $t \in [0, T]$, $\mathbf{v}(t) \in \Omega$.

Signal temporal logic (STL) [20] can express desired properties of hybrid systems. We review the syntax and robust semantics [9, 13] of STL.

Definition 1 (STL Syntax). In STL, the *atomic propositions* α and the *formulas* φ are respectively defined as follows:

$$\alpha ::= f(w_1, \dots, w_K) > 0 \quad \varphi ::= \alpha \mid \perp \mid \neg\varphi \mid \varphi \wedge \varphi \mid \square_I\varphi \mid \diamond_I\varphi \mid \varphi \mathcal{U}_I \varphi$$

Here f is a K -ary function $f : \mathbb{R}^K \rightarrow \mathbb{R}$, $w_1, \dots, w_K \in \mathbf{Var}$, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e., $I = [l, u]$, where $l, u \in \mathbb{R}$ and $l < u$. \square, \diamond and \mathcal{U} are temporal operators, which are known as *always*, *eventually* and *until*, respectively. The always operator \square and eventually operator \diamond are two special cases of the until operator \mathcal{U} , where $\diamond_I \varphi \equiv \top \mathcal{U}_I \varphi$ and $\square_I \varphi \equiv \neg \diamond_I \neg \varphi$. Other common connectives such as \vee, \rightarrow are introduced as syntactic sugar: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg \varphi_1 \wedge \neg \varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg \varphi_1 \vee \varphi_2$.

Definition 2 (STL Robust Semantics). Let \mathbf{v} be a signal, φ be an STL formula and $\tau \in \mathbb{R}_+$ be an instant. The *robustness* $R(\mathbf{v}, \varphi, \tau) \in \mathbb{R} \cup \{+\infty, -\infty\}$ of \mathbf{v} w.r.t. φ at τ is defined by induction on the construction of formulas, as follows,

$$\begin{aligned} R(\mathbf{v}, \alpha, \tau) &:= f(\mathbf{v}(\tau)) & R(\mathbf{v}, \neg \varphi, \tau) &:= -R(\mathbf{v}, \varphi, \tau) \\ R(\mathbf{v}, \varphi_1 \wedge \varphi_2, \tau) &:= \min(R(\mathbf{v}, \varphi_1, \tau), R(\mathbf{v}, \varphi_2, \tau)) \\ R(\mathbf{v}, \square_I \varphi, \tau) &:= \inf_{t \in \tau+I} R(\mathbf{v}, \varphi, t) & R(\mathbf{v}, \diamond_I \varphi, \tau) &:= \sup_{t \in \tau+I} R(\mathbf{v}, \varphi, t) \\ R(\mathbf{v}, \varphi_1 \mathcal{U}_I \varphi_2, \tau) &:= \sup_{t \in \tau+I} \min(R(\mathbf{v}, \varphi_2, t), \inf_{t' \in [\tau, t]} R(\mathbf{v}, \varphi_1, t')) \end{aligned}$$

where $\tau + [l, u]$ denotes the shifted interval $[l + \tau, u + \tau]$.

The Boolean semantics of STL, i.e., whether $(\mathbf{v}, \tau) \models \varphi$ or not, can be inferred from the quantitative robust semantics in Definition 2, namely, if $R(\mathbf{v}, \varphi, \tau) > 0$, it implies $(\mathbf{v}, \tau) \models \varphi$; and if $R(\mathbf{v}, \varphi, \tau) < 0$, it implies $(\mathbf{v}, \tau) \not\models \varphi$.

2.2 Online Robust Monitoring of STL

Online monitoring concerns the satisfaction of a *partial signal* $\mathbf{v}_{0:b} : [0, b] \rightarrow \mathbb{R}^d$ w.r.t. an STL formula φ . We define a *completion* of $\mathbf{v}_{0:b}$ as a signal $\mathbf{v} : [0, T] \rightarrow \mathbb{R}^d$ ($b \leq T$) such that $\forall t \in [0, b], \mathbf{v}(t) = \mathbf{v}_{0:b}(t)$. A completion \mathbf{v} can be written as the concatenation of $\mathbf{v}_{0:b}$ with a *suffix signal* $\mathbf{v}_{b:T}$, i.e., $\mathbf{v} = \mathbf{v}_{0:b} \cdot \mathbf{v}_{b:T}$.

Definition 3 (Online Robust Monitor [6]). Let $\mathbf{v}_{0:b}$ be a partial signal, and let φ be an STL formula. We denote by R_{\max}^α and R_{\min}^α the possible *maximum* and *minimum bounds* of the robustness $R(\mathbf{v}, \alpha, \tau)$ ¹. Then, an *online robust monitor* returns a sub-interval $[R](\mathbf{v}_{0:b}, \varphi, \tau) \subseteq [R_{\min}^\alpha, R_{\max}^\alpha]$ at instant b , which is defined as follows, by induction on the construction of formulas.

$$\begin{aligned} [R](\mathbf{v}_{0:b}, \alpha, \tau) &:= \begin{cases} [f(\mathbf{v}_{0:b}(\tau)), f(\mathbf{v}_{0:b}(\tau))] & \text{if } \tau \in [0, b] \\ [R_{\min}^\alpha, R_{\max}^\alpha] & \text{otherwise} \end{cases} \\ [R](\mathbf{v}_{0:b}, \neg \varphi, \tau) &:= -[R](\mathbf{v}_{0:b}, \varphi, \tau) \\ [R](\mathbf{v}_{0:b}, \varphi_1 \wedge \varphi_2, \tau) &:= \min([R](\mathbf{v}_{0:b}, \varphi_1, \tau), [R](\mathbf{v}_{0:b}, \varphi_2, \tau)) \\ [R](\mathbf{v}_{0:b}, \square_I \varphi, \tau) &:= \inf_{t \in \tau+I} ([R](\mathbf{v}_{0:b}, \varphi, t)) \\ [R](\mathbf{v}_{0:b}, \varphi_1 \mathcal{U}_I \varphi_2, \tau) &:= \sup_{t \in \tau+I} \min([R](\mathbf{v}_{0:b}, \varphi_2, t), \inf_{t' \in [\tau, t]} [R](\mathbf{v}_{0:b}, \varphi_1, t')) \end{aligned}$$

¹ $R(\mathbf{v}, \alpha, \tau)$ is bounded because of the bound Ω of \mathbf{v} . In practice, if Ω is unknown, we just need to set R_{\max}^α and R_{\min}^α to be ∞ and $-\infty$ respectively.

Here, f is defined as in Definition 1, and the arithmetic rules over intervals $I = [l, u]$ are defined as follows: $-I := [-u, -l]$ and $\min(I_1, I_2) := [\min(l_1, l_2), \min(u_1, u_2)]$.

We denote by $[R]^U(\mathbf{v}_{0:b}, \varphi, \tau)$ and $[R]^L(\mathbf{v}_{0:b}, \varphi, \tau)$ the upper and lower bounds of $[R](\mathbf{v}_{0:b}, \varphi, \tau)$ respectively. Intuitively, this interval $[R](\mathbf{v}_{0:b}, \varphi, \tau)$ indicates the set of robustness values possibly reached by the completion of $\mathbf{v}_{0:b}$, under any suffix signal $\mathbf{v}_{b:T}$. This interval can be used to derive a 3-valued verdict for a given $\mathbf{v}_{0:b}$, that signifies the satisfaction of $\mathbf{v}_{0:b}$ w.r.t. the specification φ : if $[R]^L(\mathbf{v}_{0:b}, \varphi, \tau) > 0$, it implies **true**, i.e., $\mathbf{v}_{0:b}$ satisfies φ ; if $[R]^U(\mathbf{v}_{0:b}, \varphi, \tau) < 0$, it implies **false**, i.e., $\mathbf{v}_{0:b}$ violates φ ; otherwise, it returns **unknown**.

3 Overview of Causation Monitoring

As mentioned in §1, the information masking issue of online robust monitors has been identified as a problem in [24, 26, 27]. The problem arises from the *monotonicity* of online robust monitors, i.e., during evolution of the signal $\mathbf{v}_{0:b}$, $[R]^U(\mathbf{v}_{0:b}, \varphi, \tau)$ monotonically decreases and $[R]^L(\mathbf{v}_{0:b}, \varphi, \tau)$ monotonically increases. The formal statement of this problem can be found in [26].

Online Causation Monitoring. *Causation monitoring* is proposed in [26] as a solution to the problem. Specifically, instead of monitoring robustness that indicates whether a partial trace violates the specification, it monitors whether the increment of the trace at each instant is the *causation* to the violation of the specification. Here, the definition of causation follows the online trace diagnostics [3, 27] that returns a (violation or satisfaction) *epoch*, which is a set of signal segments that *sufficiently* triggers the violation or satisfaction of the partial signal. Intuitively, an epoch can be considered as an explanation of a violation or satisfaction; the formal definition of epoch can be found in [3, 27]. Having the trace diagnostic result at each instant, causation monitoring aims to report such a verdict: if the current instant b of $\mathbf{v}_{0:b}$ is included in the violation epoch, it is considered as a *violation causation*; if b is included in the satisfaction epoch, it is considered as a *satisfaction causation*; otherwise, it is *irrelevant*.

The causation monitor proposed in [26] achieves this goal as follows: at each instant, it computes two quantities $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau)$ and $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau)$, that respectively indicate the distances of the current instant b from being a violation causation and a satisfaction causation. The formal definition of causation distances $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau)$ and $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau)$ are presented in Definition 4.

Definition 4 (Online Causation Monitor [26]). Let $\mathbf{v}_{0:b}$ be a partial signal and φ be an STL formula. At an instant b , an online causation monitor returns a *violation causation distance* $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau)$ and a *satisfaction causation distance* $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau)$, as defined in Table 1.

The causation verdict, regarding whether b is a causation or not, can be inferred by the results of the online causation monitor in Definition 4, as follows:

Table 1. The definitions of violation and satisfaction causation distances

| | |
|---|--|
| $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \alpha, \tau) := \begin{cases} f(\mathbf{v}_{0:b}(\tau)) & \text{if } b = \tau \\ \mathbf{R}_{\max}^\alpha & \text{otherwise} \end{cases}$ | $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \neg\varphi, \tau) := -[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau)$ |
| $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_1 \wedge \varphi_2, \tau) := \min([\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_1, \tau), [\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_2, \tau))$ | |
| $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \square_I \varphi, \tau) := \inf_{t \in \tau+I} ([\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, t))$ | |
| $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_1 \mathcal{U}_I \varphi_2, \tau) := \inf_{t \in \tau+I} \left(\max \left(\min \left(\inf_{t' \in [\tau, t]} ([\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_1, t')) \right), [\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi_2, t) \right) \right)$ | |
| $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \alpha, \tau) := \begin{cases} f(\mathbf{v}_{0:b}(\tau)) & \text{if } b = \tau \\ \mathbf{R}_{\min}^\alpha & \text{otherwise} \end{cases}$ | $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \neg\varphi, \tau) := -[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau)$ |
| $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_1 \wedge \varphi_2, \tau) := \max \left(\min([\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_1, \tau), [\mathbf{R}]^\lrcorner(\mathbf{v}_{0:b}, \varphi_2, \tau)), \min([\mathbf{R}]^\lrcorner(\mathbf{v}_{0:b}, \varphi_1, \tau), [\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_2, \tau)) \right)$ | |
| $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \square_I \varphi, \tau) := \sup_{t \in \tau+I} \left(\min([\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, t), [\mathbf{R}]^\lrcorner(\mathbf{v}_{0:b}, \square_I \varphi, \tau)) \right)$ | |
| $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_1 \mathcal{U}_I \varphi_2, \tau) := \sup_{t \in \tau+I} \left(\max \left(\min \left(\sup_{t' \in [\tau, t]} ([\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_1, t')), \inf_{t' \in [\tau, t]} [\mathbf{R}]^\lrcorner(\mathbf{v}_{0:b}, \varphi_2, t') \right), \min \left(\inf_{t' \in [\tau, t]} [\mathbf{R}]^\lrcorner(\mathbf{v}_{0:b}, \varphi_1, t'), [\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi_2, t) \right) \right) \right)$ | |

- if $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau) < 0$, then b is a violation causation;
- if $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau) > 0$, then b is a satisfaction causation;
- otherwise, i.e., $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, \tau) > 0$ and $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, \tau) < 0$, b is irrelevant.

Below, we use an example to illustrate how online causation monitor works.

Example 1. Consider the example in Fig. 1. As indicated by the robust monitor, the specification is violated by the signal after $b = 10$. By online trace diagnostics (see [27]), at $b = 19$, we can obtain a violation epoch $\{(v < 10, t) \mid t \in [5, 19]\}$, which implies that the violation so far is caused by the signal values v during $[5, 19]$. Since $b = 19$ is included in this epoch, b is then considered as a violation causation. On the other hand, we can also compute the violation causation distance $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, 0) = -1 < 0$ by Definition 4, which also indicates that b is a violation causation.

Similarly, at $b = 23$, we obtain an epoch $\{(v < 10, t) \mid t \in [5, 20]\}$, in which $b = 23$ is not included, so $b = 23$ is irrelevant. This is also shown by computing the causation distances $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, 0) = 2 > 0$ and $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \varphi, 0) = \mathbf{R}_{\min}^\alpha < 0$.

Note that the result of causation monitoring is not monotonic, e.g., while $b = 19$ is considered as a violation causation, $b = 23$ is not. This feature is clearly shown by the visualized result of causation monitoring in Fig. 1.

Relationship with Robust Monitors. As indicated by Fig. 1 and Example 1, the causation monitor is not monotonic, and thus it delivers more information

about system evolution. We refer to [26] for a more detailed explanation. Below, Lemma 1 states that the online causation monitor in Definition 4 refines the online robust monitor in Definition 3, in the sense that the monitoring results of robust monitors can be inferred from that of causation monitors. In other words, the information delivered by causation monitors is a superset of that can be delivered by classic robust monitors.

Lemma 1. The causation monitor in Definition 4 refines the classic online robust monitor in Definition 3, in the sense that the monitoring results of the robust monitor can be reconstructed from the results of the causation monitor, as follows:

$$[\mathbf{R}]^{\mathbf{U}}(\mathbf{v}_{0:b}, \varphi, \tau) = \inf_{t \in [0, b]} [\mathcal{R}]^{\ominus}(\mathbf{v}_{0:t}, \varphi, \tau), \quad [\mathbf{R}]^{\mathbf{L}}(\mathbf{v}_{0:b}, \varphi, \tau) = \sup_{t \in [0, b]} [\mathcal{R}]^{\oplus}(\mathbf{v}_{0:t}, \varphi, \tau)$$

4 Efficient Causation Monitoring

In [26], a straightforward way of synthesizing an online causation monitor has been provided that follows Definition 4. However, the synthesized monitor may not be sufficiently efficient to be deployed in practice, due to the high computational complexity when handling nested temporal operators.

Example 2. Consider the specification $\varphi \equiv \square_{[0,45]}(\diamond_{[0,5]}v < 10)$ in Fig. 1. For convenience, we name the sub-formulas of φ as follows: $\varphi' \equiv \diamond_{[0,5]}(v < 10)$, $\alpha \equiv (v < 10)$. Consider the computation of $[\mathcal{R}]^{\ominus}(\mathbf{v}_{0:b}, \varphi, 0)$ that contains nested temporal operators. According to Definition 4, we need to compute as follows:

$$\begin{aligned} [\mathcal{R}]^{\ominus}(\mathbf{v}_{0:b}, \varphi, 0) &= \inf_{t \in [0, 45]} [\mathcal{R}]^{\ominus}(\mathbf{v}_{0:b}, \varphi', t) \\ &= \inf_{t \in [0, 45]} \left(\inf_{t' \in [t, t+5]} \left(\max \left([\mathcal{R}]^{\ominus}(\mathbf{v}_{0:b}, \alpha, t'), [\mathbf{R}]^{\mathbf{U}}(\mathbf{v}_{0:b}, \alpha, t') \right) \right) \right) \end{aligned}$$

During this computation, for a fixed t' , $[\mathcal{R}]^{\ominus}(\mathbf{v}_{0:b}, \alpha, t')$ and $[\mathbf{R}]^{\mathbf{U}}(\mathbf{v}_{0:b}, \alpha, t')$ are repeatedly computed as long as it holds that $t' \in [t, t+5]$ for any $t \in [0, 45]$. This results in numerous redundant computations, which can significantly diminish the efficiency of causation monitoring.

We introduce two dynamic programming strategies, i.e., *intermediate result recording* and *sliding window*, for accelerating causation monitoring.

4.1 Intermediate Result Recording

Our efficient causation monitoring algorithm is presented in Algorithm 1. The basic idea of the algorithm is to record the intermediate monitoring results, by maintaining several worklists for each of the sub-formulas of an STL formula φ , so as to avoid redundant computations.

Algorithm 1 Efficient online causation monitoring

Require: a partial signal $\mathbf{v}_{0:b}$, an STL formula φ

- 1: **for** $\psi \in \text{SF}(\varphi)$ **do**
- 2: **for** $t \in \text{Eva}(\varphi, 0)[\psi]$ **do**
- 3: $\text{Cau}^\ominus[\psi](t) \leftarrow \mathbf{R}_{\max}^\alpha$, $\text{Cau}^\oplus[\psi](t) \leftarrow \mathbf{R}_{\min}^\alpha$
- 4: $\text{Rob}^U[\psi](t) \leftarrow \mathbf{R}_{\max}^\alpha$, $\text{Rob}^L[\psi](t) \leftarrow \mathbf{R}_{\min}^\alpha$
- 5: **for** $b \in [0, T]$ **do**
- 6: $\text{UPDATECAU}(\mathbf{v}_{0:b}, \varphi, 0)$ \triangleright monitoring at runtime
- 7: **function** $\text{UPDATECAU}(\mathbf{v}_{0:b}, \psi, \tau)$
- 8: **switch** ψ **do**
- 9: **case** α \triangleright atomic propositions
- 10: **if** $b \in \text{Eva}(\alpha)$ **then**
- 11: $\text{Cau}^\ominus[\psi](t) \leftarrow \begin{cases} f(\mathbf{v}_{0:b}(b)) & \text{if } t = b \\ +\infty & \text{otherwise} \end{cases}$
- 12: $\text{Cau}^\oplus[\psi](t) \leftarrow \begin{cases} f(\mathbf{v}_{0:b}(b)) & \text{if } t = b \\ -\infty & \text{otherwise} \end{cases}$
- 13: **case** $\neg\varphi$ \triangleright negations
- 14: $\text{UPDATECAU}(\mathbf{v}_{0:b}, \varphi, \tau)$ \triangleright recursive call
- 15: $\text{Cau}^\ominus[\psi] \leftarrow -\text{Cau}^\oplus[\varphi]$
- 16: $\text{Cau}^\oplus[\psi] \leftarrow -\text{Cau}^\ominus[\varphi]$
- 17: **case** $\varphi_1 \wedge \varphi_2$ \triangleright conjunctions
- 18: $\text{UPDATECAU}(\mathbf{v}_{0:b}, \varphi_1, \tau)$; $\text{UPDATECAU}(\mathbf{v}_{0:b}, \varphi_2, \tau)$ \triangleright recursive call
- 19: $\text{UPDATEROB}(\mathbf{v}_{0:b}, \varphi_1, \tau)$; $\text{UPDATEROB}(\mathbf{v}_{0:b}, \varphi_2, \tau)$ \triangleright see [6]
- 20: $\text{Cau}^\ominus[\psi] \leftarrow \min(\text{Cau}^\ominus[\varphi_1], \text{Cau}^\ominus[\varphi_2])$
- 21: $\text{Cau}^\oplus[\psi] \leftarrow \max(\min(\text{Cau}^\oplus[\varphi_1], \text{Rob}^L[\varphi_2]), \min(\text{Rob}^L[\varphi_1], \text{Cau}^\oplus[\varphi_2]))$
- 22: **case** $\Box_I\varphi$ \triangleright always operators
- 23: $\text{UPDATECAU}(\mathbf{v}_{0:b}, \varphi, \tau)$ \triangleright recursive call
- 24: $\text{UPDATEROB}(\mathbf{v}_{0:b}, \Box_I\varphi, \tau)$ \triangleright see [6]
- 25: $\text{Cau}^\ominus[\psi] \leftarrow \text{SLIDEMIN}(\text{Cau}^\ominus[\varphi], \text{Trans}(I))$ \triangleright call Algorithm 2
- 26: $\text{Cau}^\oplus[\psi] \leftarrow \min(\text{Rob}^L[\varphi], -\text{SLIDEMIN}(-\text{Cau}^\oplus[\varphi], \text{Trans}(I)))$

Sub-formulas and Evaluation periods. Given an STL formula φ , the sub-formula set $\text{SF}(\varphi)$ of φ is defined as follows (see an example in Example 2):

$$\begin{aligned}
 \text{SF}(\alpha) &:= \{\alpha\} & \text{SF}(\neg\varphi) &:= \{\neg\varphi\} \cup \text{SF}(\varphi) \\
 \text{SF}(\varphi_1 \wedge \varphi_2) &:= \{\varphi_1 \wedge \varphi_2\} \cup \text{SF}(\varphi_1) \cup \text{SF}(\varphi_2) & \text{SF}(\Box_I\varphi) &:= \{\Box_I\varphi\} \cup \text{SF}(\varphi)
 \end{aligned}$$

At the beginning of Algorithm 1, for each sub-formula $\psi \in \text{SF}(\varphi)$, we initialize four worklists, including $\text{Cau}^\ominus[\psi]$ and $\text{Cau}^\oplus[\psi]$ that record the violation and satisfaction causation distances, $\text{Rob}^U[\psi]$ and $\text{Rob}^L[\psi]$ that record the upper and lower robustness bounds.

Each of these four lists is defined over the *evaluation period* $\text{Eva}(\psi)$ of each ψ , that is, intuitively, the time interval that includes all the instants t such that $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \psi, t)$ is needed for the computation of $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, 0)$ (see Definition 4, and it also holds for satisfaction distances). Formally, given $\psi \in \text{SF}(\varphi)$,

the evaluation period is computed as follows. First, $Eva[\varphi, 0]$, a set that includes the evaluation periods of all the sub-formulas $\psi' \in \mathbf{SF}(\varphi)$, is computed recursively as follows:

$$\begin{aligned} Eva[\alpha, t] &:= \{\langle \alpha, t \rangle\} & Eva[\neg\varphi, t] &:= \{\langle \neg\varphi, t \rangle\} \cup Eva[\varphi, t] \\ Eva[\varphi_1 \wedge \varphi_2, t] &:= \{\langle \varphi_1 \wedge \varphi_2, t \rangle\} \cup Eva[\varphi_1, t] \cup Eva[\varphi_2, t] \\ Eva[\Box_I\varphi, t] &:= \{\langle \Box_I\varphi, t \rangle\} \cup \bigcup_{t' \in t+I} Eva[\varphi, t'] \end{aligned}$$

Then, the evaluation period $Eva[\varphi, 0](\psi)$ (we denote as $Eva(\psi)$ for simplicity) of ψ is defined as $Eva(\psi) = \{t \mid \langle \psi, t \rangle \in Eva[\varphi, 0]\}$.

Example 3. Consider the sub-formulas φ , φ' and α of the formula φ in Example 2. The evaluation periods of these sub-formulas can be computed as follows.

- First, $Eva[\varphi, 0] = \{\langle \varphi, 0 \rangle\} \cup \bigcup_{t' \in [0, 45]} \{\langle \varphi', t' \rangle\} \cup \bigcup_{t'' \in [0, 50]} \{\langle \varphi'', t'' \rangle\}$;
- Then, we can obtain the evaluation periods for φ , φ' and α , respectively as follows: $Eva(\varphi) = 0$, $Eva(\varphi') = [0, 45]$, $Eva(\alpha) = [0, 50]$.

Monitoring Algorithm. During the growth of partial signal $\mathbf{v}_{0:b}$, Algorithm 1 monitors $\mathbf{v}_{0:b}$ by calling the function UPDATECAU at each instant, which updates the worklists $\mathbf{Cau}^\ominus[\psi]$ and $\mathbf{Cau}^\oplus[\psi]$, such that $\mathbf{Cau}^\ominus[\psi](t)$ equals to $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \psi, t)$ and $\mathbf{Cau}^\oplus[\psi](t)$ equals to $[\mathcal{R}]^\oplus(\mathbf{v}_{0:b}, \psi, t)$, as defined in Definition 4. Unlike the plain monitoring algorithm in Example 2, when updating $\mathbf{Cau}^\ominus[\psi]$ and $\mathbf{Cau}^\oplus[\psi]$, Algorithm 1 relies on the worklists of the sub-formulas of ψ that are already available rather than computing the causation distances of the sub-formulas from scratch, thereby saving monitoring time significantly. We illustrate this process in Example 4.

As shown in Algorithm 1, UPDATECAU is defined recursively based on the structure of an STL formula. In Algorithm 1, we only show a part of the operators; other operators can be derived by the STL syntax (Definition 1) and the presented operators.

- The updates for α and $\neg\varphi$ exactly follow Definition 4;
- The update for $\varphi_1 \wedge \varphi_2$ requires not only the worklists of causation distances of sub-formulas, but also the worklists of robustness bounds of sub-formulas (according to Definition 4), so it calls the auxiliary function UPDATEROB (see Algorithm 2 in [6]) to update the worklists $\mathbf{Rob}^\ominus[\varphi_1]$ and $\mathbf{Rob}^\ominus[\varphi_2]$ of robustness bounds of sub-formulas. The function UPDATEROB was originally introduced in [6]. It updates the worklists of robustness bounds in a similar way to what UPDATECAU does for causation distances, i.e., when updating the worklists of robustness bounds for a formula ψ , it also relies on the worklists of robustness bounds for its sub-formulas, rather than computing from scratch.
- The update for $\Box_I\varphi$ requires to compute the minimum of $\mathbf{Cau}^\ominus[\varphi]$ over the time window $t + I$, for each $t \in Eva[\Box_I\varphi]$. To efficiently update the list, we adapt a sliding window algorithm [18] (elaborated on in §4.2) that, given a list

Algorithm 2 Sliding window algorithm

Require: a list $A = \{a_1, \dots, a_N\}$, a window $\omega = [l, u]$ ($l, u \in \mathbb{N}$)

Ensure: a list $\text{result} = \{\min_{j \in [i+l, i+u]} a_j \mid i \in \{1, \dots, N - u\}\}$

```

1: function SLIDEMIN( $A, \omega$ )
2:    $Q \leftarrow$  empty double-ended queue
3:    $\text{result} \leftarrow \emptyset$  ▷ initialize the list of results
4:   PUSHBACK( $Q, l + 1$ )
5:   for  $i \in \{l + 2, \dots, N\}$  do
6:     if  $i \geq u + 1$  then ▷ should give outputs
7:        $\text{result} \leftarrow \text{result} \cup \{a_{\text{FRONT}(Q)}\}$  ▷ record result
8:       if  $a_i < a_{i-1}$  then ▷ the back is not possible in result
9:         POPBACK( $Q$ ) ▷ remove back
10:        while  $a_i < a_{\text{BACK}(Q)}$  do ▷ recursive check
11:          POPBACK( $Q$ ) ▷ remove back
12:          PUSHBACK( $Q, i$ ) ▷  $a_i$  possibly in result
13:          if  $i > \text{FRONT}(Q) + u - l$  then ▷ front is no more in window
14:            POPFRONT( $Q$ ) ▷ remove front
15:   return result
    
```

$\{a_1, \dots, a_N\}$ and an index window $[l, u]$ ($l, u \in \mathbb{N}$), computes the minimum $\min_{j \in [i+l, i+u]} a_j$ over the window $[i + l, i + u]$ for each $i \in \{1, \dots, N - u\}$. In Algorithm 1, $\text{Trans}(I)$ transforms a time interval I to the index representation of a window, simply by considering the sampling frequency².

4.2 Sliding Window Algorithm

The sliding window algorithm [18] is given in Algorithm 2, which computes the local minimum $\min_{j \in [i+l, i+u]} a_j$ over the span $[i + l, i + u]$, for each $i \in \{1, \dots, N - u\}$. This is yet another dynamic programming strategy, by using a double-ended queue Q (Line 2) to record the comparisons that have been performed between the elements in Q , and thus eliminate redundant comparisons.

Algorithm 2 takes as input a list $\{a_1, \dots, a_N\}$ and a window $[l, u]$. Initially, the window is placed to contain the elements a_{1+l}, \dots, a_{1+u} , and the index $l + 1$ is pushed to the back of Q (Line 4). Then it enters a loop to traverse the list (Line 5). Inside the loop, first, it checks the condition whether a result should be reported, i.e., the elements in the initial window have been traversed (Line 6). From which that on, it reports the list element with the index at the front of Q at each loop (Line 7). Then, it recursively removes the indexes $a_{\text{BACK}(Q)}$ at the back of Q , if the element a_i of the current index i is greater than the $a_{\text{BACK}(Q)}$ (Line 8-11), and pushes the current index i to the back of Q (Line 12). If the

² In practice, the continuous time domain of signals (see §2.1) needs to be discretized, by sampling the signal with a certain frequency. In this way, a signal can be represented as a list, which is the format required in Algorithm 2.

index at the front of Q is already out of the scope of the window, then it is also removed (Line 14).

Note that, the index of local minimum over the window is always stored at the front of Q , and that is why it is returned at each loop in Line 7 of Algorithm 2. In this way, the comparisons that have been performed between list elements are reflected by the state of Q , thus reducing redundancies significantly.

Example 4. Consider the specification in Example 2. According to Algorithm 1, our computation of $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, 0)$ at $b = 19$ relies on updating the worklists for the sub-formulas $\text{SF}(\varphi)$ of φ . As shown in Table 2, we need to maintain five worklists for the sub-formulas of φ . Each worklist is defined over the evaluation period of the sub-formula, as computed in Example 3.

Due to recursive call of `UPDATECAU` and `UPDATEROB`, our algorithm first updates the worklist $\text{Cau}^\ominus[\alpha]$ and $\text{Rob}^\text{U}[\alpha]$ for α , as shown by the corresponding rows in Table 2. Then, $\text{Rob}^\text{U}[\varphi']$ is updated by taking the local maximum over each window $[0, 5]$ (by `UPDATEROB`), and $\text{Cau}^\ominus[\varphi']$ is updated based on $\text{Cau}^\ominus[\alpha]$ and $\text{Rob}^\text{U}[\varphi']$ (by Algorithm 1). Finally, $\text{Cau}^\ominus[\varphi]$ can be updated based on $\text{Cau}^\ominus[\varphi']$.

Table 2. The worklists for computing $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \varphi, 0)$ at $b = 19$

| b (time) | 0 | ... | 5 | ... | 10 | ... | 14 | 15 | ... | 19 | 20 | ... | 45 | ... | 50 |
|---------------------------------|-------------------|-----|-------------------|-----|-------------------|-----|-------------------|-------------------|-----|-------------------|-------------------|-----|-------------------|-----|-------------------|
| $\text{Cau}^\ominus[\alpha]$ | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α | R_{\max}^α | ... | -1 | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α |
| $\text{Rob}^\text{U}[\alpha]$ | 10 | ... | 0 | ... | -4 | ... | -3.5 | -3 | ... | -1 | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α |
| $\text{Rob}^\text{U}[\varphi']$ | 10 | ... | 0 | ... | -3 | ... | -1 | R_{\max}^α | ... | R_{\max}^α | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α |
| $\text{Cau}^\ominus[\varphi']$ | R_{\max}^α | ... | R_{\max}^α | | R_{\max}^α | | -1 | R_{\max}^α | | R_{\max}^α | R_{\max}^α | ... | R_{\max}^α | ... | R_{\max}^α |
| $\text{Cau}^\ominus[\varphi]$ | -1 | | | | | | | | | | | | | | |

Compare our algorithm with the naive one in Example 2. In our algorithm, the computations of $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \alpha, t)$ and $[\text{R}]^\text{U}(\mathbf{v}_{0:b}, \alpha, t)$ for any specific t both happen only once; then, they are recorded in the worklists and when they are used to update the worklists of other formulas, they can be directly read from the worklist, which does not take new computation costs. Therefore, our algorithm avoids the repeated computation of $[\mathcal{R}]^\ominus(\mathbf{v}_{0:b}, \alpha, t)$ and $[\text{R}]^\text{U}(\mathbf{v}_{0:b}, \alpha, t)$, as shown in the monitoring process in Example 2.

5 Demonstration of CauMon

We implemented CauMon in C++, which can be easily compiled to interface with CPS implemented in any formalism. In this section, we showcase the usage of CauMon, by compiling it to be a MATLAB API, based on the MEX functions of MATLAB. A code snippet for monitoring φ in Fig. 1 is shown as follows.

```

1 signal = 'speed';
2 spec = 'alw_[0,45](ev_[0,5](speed[t]<10))';
3 tau = 0;
4 while ~end()
5     trace = get_trace();
6     [vio_d, sat_d] = cau_mon(signal, trace, spec, tau);
7 end

```

The function `cau_mon` serves as the interface to call our causation monitor, which requires four arguments, namely, a trace, a list of signal names, an STL specification and a non-negative τ . Their formats are required as follows:

- `trace` is a high-dimensional array. Its first row is an array of time stamps; from second row on, each row denotes a signal concerned with the STL;
- `signal` is a string that denotes a list of signals that correspond to each row of `trace`. Multiple signals can be separated by commas.
- `spec` is a string that denotes an STL formula. The syntax of `spec` follows the standard in Breach [8].
- `tau` is a non-negative real, as τ defined in Definition 4.

During the monitoring of system execution, the program iteratively calls `cau_mon` with an updated `trace`, which is obtained by a function `get_trace`. The function `get_trace` requires an interface with the system being monitored. Typically, such an interface is provided by a CPS simulator; for example, in the case of Simulink models, one can use the function `sim` to obtain the output of Simulink models. At each instant, `cau_mon` can return a violation causation distance `vio_d` and a satisfaction causation distance `sat_d`, exactly as defined in Definition 4.

6 Experimental Evaluation

We introduce the experimental evaluation of our tool CauMon. For the purpose of comparison, we also integrated two baseline monitors, namely, RobM (the online robust monitor from [6]) and PCauM (the plain implementation of online causation monitor from Definition 4) into our tool, as an option that can be specified by users. Our tool is publicly available in our Github repository³.

6.1 Experiment Setting

Benchmarks. To evaluate the efficiency of our proposed monitoring algorithm, we collect traces from four MATLAB Simulink models that are commonly-used in the CPS community [10–12, 21].

Automatic Transmission (AT) implements a transmission controller of an automotive system. It has been widely-used recently [10–12] as a benchmark of CPS testing and monitoring. It contains 64 blocks in total, including a *stateflow* chart

³ <https://github.com/choshina/EfficientCausationMonitor>.

that represents the transmission control logic. The outputs of AT, including **speed** and **RPM**, reflect the state of the automotive system. The specifications that AT are expected to hold are listed as follows:

- $\varphi_1^{AT} \equiv \square_{[0,30]}(\text{speed} < 110)$: **speed** should be always low;
- $\varphi_2^{AT} \equiv \square_{[0,29]}(\text{speed} > 70 \rightarrow \diamond_{[0,1]}(\text{speed} > 80))$: there should be a drastic **speed** change from 70 to 80;
- $\varphi_3^{AT} \equiv \square_{[0,27]}(\text{speed} > 50 \rightarrow \diamond_{[1,3]}(\text{RPM} < 3000))$: whenever **speed** is higher than 50, **RPM** should be below 3000 in 3s;
- $\varphi_4^{AT} \equiv \square_{[0,29]}(\text{speed} < 100) \vee \square_{[29,30]}(\text{speed} > 65)$: there should not be a drastic **speed** change at the end of the simulation;

Abstract Fuel Control (AFC) is a powertrain control system released by Toyota [17] and has been widely used as a benchmark in CPS community [10–12, 21]. The system takes external inputs including engine speed and pedal angle, and adjusts the *air-to-fuel ratio* to ensure the performance of the powertrain system. The output of AFC includes the *air-to-fuel ratio* **AF** and a reference value **AFref**. The specifications of AFC are listed as follows:

- $\varphi_1^{AFC} \equiv \square_{[10,50]}(|\text{AF} - \text{AFref}| < 0.1)$: the deviation of **AF** from **AFref** should always be small;
- $\varphi_2^{AFC} \equiv \square_{[10,48.5]} \diamond_{[0,1.5]} (|\text{AF} - \text{AFref}| < 0.08)$: a large deviation should not last for too long;

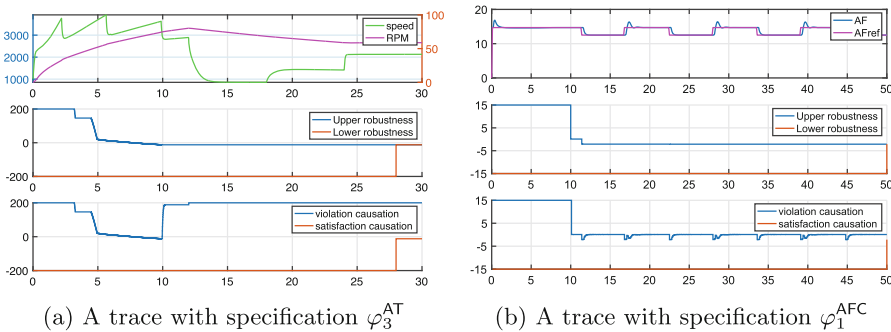


Fig. 2. Comparison between CauMon and RobM, in terms of the information provided by different monitors. In each of the sub-figures, the top plot is the signals, the middle plot is the result of online robust monitors [6], and the bottom plot is the result of online causation monitors.

Neural Network Controller (NN). This is a magnetic levitation system that has been used as a benchmark in [10, 12, 21, 29]. It takes one input signal, $Ref \in [1, 3]$, which is the reference for the position Pos of a magnet suspended above an electromagnet. The specification of NN is a complicated one:

- $\varphi_1^{\text{NN}} \equiv \square_{[0,18]}(\neg \text{close} \rightarrow \text{reach})$, where $\text{close} \equiv |Pos - Ref| \leq \rho + a \cdot |Ref|$, and $\text{reach} \equiv \diamond_{[0,2]}(\square_{[0,1]}(\text{close}))$: the position should approach the reference position in some seconds when they are far. Here, $a = 0.04$ and $\rho = 0.004$.

Free Floating Robot (FFR) models a robot moving in a 2D space [28, 30]. It takes as input the four boosters of the robot, and outputs four signals that are the position in terms of coordinate values x, y . The specification of FFR is as follows:

- $\varphi_1^{\text{FFR}} \equiv \neg(\diamond_{[0,5]}(\square_{[0,2]}(x \in [1.5, 1.7] \wedge y \in [1.5, 1.7])))$: it requires the robot not to stay in an area for at least 2 s.

Experiment Design. For each specification, we first generate 10 signals by running the Simulink models with random inputs, and then for each signal, we apply the three monitors and compare the total time they spent in monitoring the signal. To handle the fluctuation of monitoring time due to the environmental noises, we repeat each monitoring process for five times and report the average monitoring time as the result. While some monitors may be not efficient and not terminated within a reasonable time budget, we set 5000 s as a timeout.

Our experiments are conducted on Amazon EC2 c4.2xlarge instances (2.9 GHz Intel Xeon E5-2666 v3, 15 GB RAM).

6.2 Evaluation

Efficacy of CauMon. To demonstrate the superiority of CauMon compared to RobM in terms of informativeness, we show two plots in Fig. 2, which depict the signals produced by Simulink models and the monitoring results of CauMon and RobM. Due to the page limit, examples of other specifications are presented in our Github repository. We can observe that, while the upper robustness curves in these three plots provided by RobM are always monotonic, the violation causation distances provided by CauMon are not monotonic, thus they can deliver more information about system evolution. For instance, in Fig. 2b, the spikes shown in the monitoring result of CauMon reflect that the deviation between AF and AFref is greater than the threshold 0.1 in φ_1^{AFC} for more than once. However, this information can not be provided by the monotonic curve of the upper robustness from the monitoring result given by RobM.

Efficiency of CauMon. The experimental results are presented in Table 3. Each sub-table reports the results of monitoring 10 signals for the corresponding specification. The first three columns of each sub-table report the total monitoring time of the three monitors, and the last two columns report the comparison between the proposed CauMon and the two baseline monitors RobM and CauMon, computed by $\Delta A = (\text{CauMon} - A)/A$, where A is either PCauM or RobM.

First, by the comparison between PCauM and CauMon, we observe that in our experiments, CauMon always outperforms PCauM, with an improvement of at least 17.5% (in φ_4^{AT}). In particular, in those complex specifications that have nested temporal operators, such as φ_2^{AT} , φ_3^{AT} and φ_1^{NN} , PCauM can take extremely

long time to monitor the traces: for φ_2^{AT} and φ_3^{AT} , it takes around 1900s; for φ_1^{NN} that has nested operators of three levels, it even gets timeout. In the case of φ_2^{AT} and φ_3^{AT} , each instant of the signals takes about $\frac{1900}{30 \times 100} = 0.63$ seconds, which is 63 times longer than the sampling period (0.01s for AT) of the traces. This performance can lead to severe delays if PCauM is deployed in practice, and the main reason is, as shown in Example 2, because of the redundant computations of intermediate results. In contrast, CauMon does not suffer from the problem and provides a monitoring performance that allows its usage in a real-world setting, possibly even a synchronous monitoring setting.

The performance of PCauM is severely subject to the complexity of the specification. In the problems that have simpler formulas (e.g., φ_1^{AT} , φ_1^{AFC}), PCauM is not very slow. However, for specifications that have nested operators (e.g., φ_2^{AT} , φ_3^{AT} , φ_1^{NN}), PCauM becomes not feasible in practice. By contrast, CauMon suffers much less from this issue. Even for complex specifications, CauMon is still very efficient and its performance is always comparable with RobM.

By the comparison between RobM and CauMon, we observe that, the performance of CauMon is at the same magnitude of RobM, and so its performance

Table 3. Experimental results of the three monitors RobM, PCauM and CauMon Time is reported in seconds.

| φ_1^{AT} | RobM | PCauM | CauMon | CauMon stat. (%) | | φ_2^{AT} | RobM | PCauM | CauMon | CauMon stat. (%) | | φ_3^{AT} | RobM | PCauM | CauMon | CauMon stat. (%) | |
|-------------------------|------|-------|--------|------------------|----------------|--------------------------|--------|---------|--------|------------------|---------------|--------------------------|----------------|---------|--------|------------------|-------|
| | | | | Δ RobM | Δ PCauM | | | | | | Δ RobM | | Δ PCauM | | | | |
| #1 | 0.34 | 0.31 | 0.20 | -41.8 | -35.5 | #1 | 1.55 | 1972.82 | 1.99 | +28.4 | -99.9 | #1 | 1.63 | 2007.99 | 2.26 | +39.1 | -99.9 |
| #2 | 0.32 | 0.31 | 0.19 | -40.2 | -38.7 | #2 | 1.50 | 1907.76 | 1.87 | +24.9 | -99.9 | #2 | 1.64 | 2015.69 | 2.27 | +38.2 | -99.9 |
| #3 | 0.30 | 0.3 | 0.18 | -40.4 | -40.0 | #3 | 1.47 | 1932.85 | 1.87 | +27.5 | -99.9 | #3 | 1.63 | 1990.24 | 2.13 | +30.5 | -99.9 |
| #4 | 0.33 | 0.31 | 0.19 | -40.6 | -38.7 | #4 | 1.48 | 1914.46 | 1.84 | +24.8 | -99.9 | #4 | 1.59 | 1926.94 | 2.20 | +38.2 | -99.9 |
| #5 | 0.39 | 0.34 | 0.23 | -41.6 | -32.4 | #5 | 1.50 | 1902.60 | 1.88 | +25.8 | -99.9 | #5 | 1.62 | 1985.16 | 2.20 | +35.8 | -99.9 |
| #6 | 0.30 | 0.30 | 0.18 | -40.1 | -40.0 | #6 | 1.48 | 1887.34 | 1.86 | +25.7 | -99.9 | #6 | 1.64 | 2023.44 | 2.27 | +38.4 | -99.9 |
| #7 | 0.32 | 0.31 | 0.19 | -40.7 | -38.7 | #7 | 1.45 | 1891.06 | 1.82 | +25.7 | -99.9 | #7 | 1.57 | 1979.15 | 1.99 | +27.0 | -99.9 |
| #8 | 0.34 | 0.32 | 0.20 | -41.0 | -37.5 | #8 | 1.44 | 1847.76 | 1.82 | +26.2 | -99.9 | #8 | 1.57 | 1894.74 | 2.16 | +37.9 | -99.9 |
| #9 | 0.31 | 0.31 | 0.19 | -39.9 | -38.7 | #9 | 1.58 | 1865.24 | 1.91 | +21.0 | -99.9 | #9 | 1.63 | 1991.59 | 2.11 | +29.2 | -99.9 |
| #10 | 0.33 | 0.32 | 0.20 | -40.9 | -37.5 | #10 | 1.41 | 1855.63 | 1.81 | +28.2 | -99.9 | #10 | 1.61 | 1997.21 | 2.16 | +33.9 | -99.9 |
| φ_1^{AT} | RobM | PCauM | CauMon | CauMon stat. (%) | | φ_1^{AFC} | RobM | PCauM | CauMon | CauMon stat. (%) | | φ_2^{AFC} | RobM | PCauM | CauMon | CauMon stat. (%) | |
| | | | | Δ RobM | Δ PCauM | | | | | | Δ RobM | | Δ PCauM | | | | |
| #1 | 0.46 | 0.57 | 0.47 | +1.5 | -17.5 | #1 | 0.0079 | 0.0095 | 0.0056 | -29.8 | -41.1 | #1 | 0.0084 | 0.7033 | 0.0087 | +3.1 | -98.8 |
| #2 | 0.41 | 0.53 | 0.42 | +2.4 | -20.8 | #2 | 0.0066 | 0.0088 | 0.0048 | -27.1 | -45.5 | #2 | 0.0072 | 0.7041 | 0.0078 | +8.0 | -98.9 |
| #3 | 0.45 | 0.57 | 0.46 | +2.2 | -19.3 | #3 | 0.0063 | 0.0086 | 0.0047 | -25.1 | -45.3 | #3 | 0.0069 | 0.7017 | 0.0077 | +12.0 | -98.9 |
| #4 | 0.41 | 0.53 | 0.42 | +2.3 | -20.8 | #4 | 0.0061 | 0.0086 | 0.0047 | -23.3 | -45.3 | #4 | 0.0066 | 0.7041 | 0.0076 | +15.0 | -98.9 |
| #5 | 0.43 | 0.55 | 0.44 | +2.1 | -20.0 | #5 | 0.0061 | 0.0086 | 0.0047 | -23.1 | -45.3 | #5 | 0.0067 | 0.702 | 0.0077 | +15.1 | -98.9 |
| #6 | 0.44 | 0.56 | 0.45 | +1.4 | -19.6 | #6 | 0.0061 | 0.0086 | 0.0046 | -23.8 | -46.5 | #6 | 0.0066 | 0.7058 | 0.0076 | +15.0 | -98.9 |
| #7 | 0.42 | 0.54 | 0.43 | +1.1 | -20.4 | #7 | 0.0061 | 0.0086 | 0.0047 | -23.7 | -45.3 | #7 | 0.0067 | 0.7041 | 0.0077 | +14.8 | -98.9 |
| #8 | 0.41 | 0.54 | 0.42 | +1.4 | -22.2 | #8 | 0.0061 | 0.0086 | 0.0047 | -23.6 | -45.3 | #8 | 0.0067 | 0.7053 | 0.0077 | +16.1 | -98.9 |
| #9 | 0.43 | 0.55 | 0.43 | +1.0 | -21.8 | #9 | 0.0061 | 0.0086 | 0.0047 | -23.6 | -45.3 | #9 | 0.0067 | 0.7052 | 0.0077 | +15.4 | -98.9 |
| #10 | 0.46 | 0.57 | 0.46 | +0.5 | -19.3 | #10 | 0.0062 | 0.0087 | 0.0047 | -24.1 | -46.0 | #10 | 0.0067 | 0.7051 | 0.0078 | +15.8 | -98.9 |
| φ_1^{NN} | RobM | PCauM | CauMon | CauMon stat. (%) | | φ_1^{FER} | RobM | PCauM | CauMon | CauMon stat. (%) | | | | | | | |
| | | | | Δ RobM | Δ PCauM | | | | | | Δ RobM | Δ PCauM | | | | | |
| #1 | 1.25 | t/o | 1.89 | +51.4 | -99.9 | #1 | 0.053 | 850.1 | 0.085 | +60.7 | -99.9 | | | | | | |
| #2 | 1.22 | t/o | 1.86 | +52.8 | -99.9 | #2 | 0.049 | 857.6 | 0.080 | +62.3 | -99.9 | | | | | | |
| #3 | 1.23 | t/o | 1.89 | +53.4 | -99.9 | #3 | 0.049 | 857.1 | 0.080 | +63.0 | -99.9 | | | | | | |
| #4 | 1.22 | t/o | 1.88 | +53.7 | -99.9 | #4 | 0.051 | 822.0 | 0.082 | +61.7 | -99.9 | | | | | | |
| #5 | 1.23 | t/o | 1.88 | +52.7 | -99.9 | #5 | 0.049 | 813.5 | 0.080 | +62.8 | -99.9 | | | | | | |
| #6 | 1.22 | t/o | 1.88 | +53.8 | -99.9 | #6 | 0.050 | 822.0 | 0.081 | +63.3 | -99.9 | | | | | | |
| #7 | 1.23 | t/o | 1.88 | +53.0 | -99.9 | #7 | 0.051 | 867.8 | 0.083 | +61.4 | -99.9 | | | | | | |
| #8 | 1.24 | t/o | 1.89 | +52.8 | -99.9 | #8 | 0.047 | 809.0 | 0.077 | +62.3 | -99.9 | | | | | | |
| #9 | 1.24 | t/o | 1.88 | +52.3 | -99.9 | #9 | 0.047 | 809.6 | 0.077 | +64.0 | -99.9 | | | | | | |
| #10 | 1.23 | t/o | 1.86 | +51.4 | -99.9 | #10 | 0.047 | 809.4 | 0.077 | +63.1 | -99.9 | | | | | | |

is comparable with RobM. While in some cases CauMon is not as fast as RobM, the performance difference is not very large. This is acceptable, regarding that our CauMon can provide more information about system evolution than RobM. There also happens that CauMon is faster than RobM, with simple specifications that have no nested temporal operators, such as φ_1^{AT} and φ_1^{AFC} . This is because monitoring simple specifications, like $\Box_I \alpha$, mainly needs the causation distance lists $\text{Cau}^\ominus[\alpha]$ and $\text{Cau}^\oplus[\alpha]$ of atomic proposition α , and by Defs. 3 and 4, $\text{Cau}^\ominus[\alpha]$ and $\text{Cau}^\oplus[\alpha]$ have simpler shape than $\text{Rob}^{\text{U}}[\alpha]$ and $\text{Rob}^{\text{L}}[\alpha]$.

7 Conclusion and Future Work

We propose an efficient approach for online causation monitoring. Our approach features two dynamic programming strategies, namely, the use of causation distance lists that record intermediate results, and the use of sliding window algorithms that accelerate the causation computation of temporal operators. Experiments show that, in terms of efficiency, our approach significantly outperforms the plain causation monitor in [26], and is comparable with the existing online robust monitors that deliver less information about system evolution than ours.

As future work, we would like to explore the application of the proposed monitors for system behavior analysis. For instance, by causation monitoring, we can obtain the information about when a cause of the specification violation happens, and this can be used for fault analysis such as localization and repair.

Acknowledgments. Z. Zhang is supported by JSPS KAKENHI Grant No. JP23K16865 and No. JP23H03372. P. Arcaini is supported by Engineerable AI Techniques for Practical Applications of High-Quality Machine Learning-based Systems Project (Grant Number JPMJMI20B8), JST-Mirai. J. An and I. Hasuo are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPM-JER1603), JST, Funding Reference number: 10.13039/501100009024 ERATO.

Data Availability Statement. All relevant data that support the findings of this paper are available in Zenodo with the identifier <https://doi.org/10.5281/zenodo.12518433>.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TALIRO: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
2. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification. LNCS, vol. 10457. Springer, Cham (2018). <https://doi.org/10.1007/978-3-319-75632-5>
3. Bartocci, E., Ferrère, T., Manjunath, N., Ničković, D.: Localizing faults in Simulink/Stateflow models with STL. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), pp. 197–206. HSCC 2018, Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3178126.3178131>
4. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006). https://doi.org/10.1007/11944836_25
5. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 1–64 (2011). <https://doi.org/10.1145/2000799.2000800>
6. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods Syst. Des. **51**(1), 5–30 (2017). <https://doi.org/10.1007/s10703-017-0286-7>
7. Dokhanchi, A., Hoxha, B., Fainekos, G.: On-line monitoring for temporal logic robustness. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 231–246. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_19
8. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
9. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
10. Ernst, G., et al.: ARCH-COMP 2021 category report: falsification with validation of results. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21). EPiC Series in Computing, vol. 80, pp. 133–152. EasyChair (2021). <https://doi.org/10.29007/xw11>
11. Ernst, G., et al.: ARCH-COMP 2020 category report: falsification. In: Frehse, G., Althoff, M. (eds.) 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). EPiC Series in Computing, vol. 74, pp. 140–152. EasyChair (2020). <https://doi.org/10.29007/trr1>
12. Ernst, G., et al.: ARCH-COMP 2022 category report: falsification with Unbounded resources. In: Frehse, G., Althoff, M., Schoitsch, E., Guiochet, J. (eds.) Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 204–221. EasyChair (2022). <https://doi.org/10.29007/fhnk>
13. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoret. Comput. Sci. **410**(42), 4262–4291 (2009). <https://doi.org/10.1016/j.tcs.2009.06.021>

14. Ho, H.-M., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 178–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_15
15. Jakšić, S., Bartocci, E., Grosu, R., Kloibhofer, R., Nguyen, T., Ničković, D.: From signal temporal logic to FPGA monitors. In: Proceedings of the 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign, pp. 218–227. MEMOCODE 2015, IEEE Computer Society, USA (2015). <https://doi.org/10.1109/MEMCOD.2015.7340489>
16. Jakšić, S., Bartocci, E., Grosu, R., Nguyen, T., Ničković, D.: Quantitative monitoring of STL with edit distance. *Formal Methods Syst. Des.* **53**, 83–112 (2018). <https://doi.org/10.1007/s10703-018-0319-x>
17. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, pp. 253–262. HSCC 2014, ACM, New York, NY, USA (2014). <https://doi.org/10.1145/2562059.2562140>
18. Lemire, D.: Streaming maximum-minimum filter using no more than three comparisons per element. *Nordic J. Comput.* **13**(4), 328–339 (2006)
19. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: Explainable online monitoring of metric temporal logic. In: Sankaranarayanan, S., Sharygina, N. (eds.) International Conference on Tools and Algorithms for the Construction and Analysis of Systems, vol. 13994, pp. 473–491. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30820-8_28
20. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
21. Menghi, C., et al.: ARCH-COMP23 category report: Falsification. In: Frehse, G., Althoff, M. (eds.) Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, vol. 96, pp. 151–169. EasyChair (2023). <https://doi.org/10.29007/6nqs>
22. Nickovic, D., Maler, O.: AMT: A property-based monitoring tool for analog systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75454-1_22
23. Ničković, D., Yamaguchi, T.: RTAMT: online robustness monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 564–571. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_34
24. Selyunin, K., et al.: Runtime monitoring with recovery of the SENT communication protocol. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 336–355. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_17
25. Ulus, D.: Online monitoring of metric temporal logic using sequential networks. arXiv preprint [arXiv:1901.00175](https://arxiv.org/abs/1901.00175) (2019)
26. Zhang, Z., An, J., Arcaini, P., Hasuo, I.: Online causation monitoring of signal temporal logic. In: Enea, C., Lal, A. (eds.) Computer Aided Verification, pp. 62–84. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37706-8_4
27. Zhang, Z., Arcaini, P., Xie, X.: Online reset for signal temporal logic monitoring. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **41**(11), 4421–4432 (2022). <https://doi.org/10.1109/TCAD.2022.3197693>

28. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by monte Carlo tree search. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **37**(11), 2894–2905 (2018)
29. Zhang, Z., Hasuo, I., Arcaini, P.: Multi-armed bandits for Boolean connectives in hybrid system falsification. In: Dillig, I., Tasiran, S. (eds.) *CAV 2019*. LNCS, vol. 11561, pp. 401–420. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_23
30. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using monte Carlo tree search guided by QB-robustness. In: Silva, A., Leino, K.R.M. (eds.) *CAV 2021*. LNCS, vol. 12759, pp. 595–618. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_29

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

