Runtime Enforcement of CPS against Signal Temporal Logic

Han Su
suhan@ios.ac.cn
Institute of Software, Chinese Academy of Sciences
& University of Chinese Academy of Sciences
Beijing, China

Saumya Shankar*
Saumya.shankar@auckland.ac.nz
Department of Electrical, Computer and Software Engineering
The University of Auckland
Auckland, New Zealand

Srinivas Pinisetty
spinisetty@iitbbs.ac.in
School of Electrical and Computer Sciences
Indian Institute of Technology Bhubaneswar
Bhubaneswar, India

Partha S. Roop*
p.roop@auckland.ac.nz
Department of Electrical, Computer and Software Engineering
The University of Auckland
Auckland, New Zealand

Naijun Zhan*
znj@ios.ac.cn
Peking University
School of Computer Science
Key Lab. of High Confidence Software Technology
Beijing, China

ABSTRACT

Cyber-Physical Systems (CPSs), especially those involving autonomy, need guarantees of their safety. Runtime Enforcement (RE) is a lightweight method to formally ensure that some specified properties are satisfied over the executions of the system. Hence, there is recent interest in the RE of CPS. However, existing methods are not designed to tackle specifications suitable for the hybrid dynamics of CPS. With this in mind, we develop runtime enforcement of CPS using properties defined in Signal Temporal Logic (STL).

In this work, we aim to construct a runtime enforcer for a given STL formula to minimally modify a signal to satisfy the formula. To achieve this, the STL formula to be enforced is first translated into a timed transducer, while the signal to be corrected is encoded as timed words. We provide timed transducers for the temporal operators *until* and *release* noting that other temporal operators can be expressed using these two. Our approach enables effective enforcement of STL properties for CPS. A case study is provided to illustrate the approach and generate empirical evidence of its suitability for CPS.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Formal methods and theory of security**; • **Theory of computation → Transducers**.

KEYWORDS

Reactive System, Runtime Enforcement, Signal Temporal Logic, Timed Automata, Timed Transducer

*The corresponding authors

ACM Reference Format:
Han Su, Saumya Shankar, Srinivas Pinisetty, Partha S. Roop, and Naijun Zhan. 2025. Runtime Enforcement of CPS against Signal Temporal Logic. In *28th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3716863.3718052

1 INTRODUCTION

Modern CPSs, especially those with emerging AI-enabled modules, are becoming increasingly difficult to verify formally, and sometimes even impossible, due to the chaotic behavior of the AI modules. Runtime Enforcement [30], serving as a lightweight formal method, has attracted increasing research interest in recent years for the verification of CPSs. In RE, an *enforcer* is synthesized to monitor the executions of a black-box system at runtime, ensuring compliance with a set of desired properties. In the event of a violation, the enforcer employs evasive actions to output property complaint words. There are various evasive actions, such as: (i) blocking the execution [30], (ii) modifying the input sequence by suppressing and/or inserting actions [14, 15], and (iii) buffering input actions until they can be safely forwarded [8–10, 22, 23]. However, these evasive actions may not be suitable for CPSs since delaying reactions or terminating the system may be impractical [25].

One key aspect of CPS is the need for active interaction between the controller (the Cyber part) and the plant (the Physical part) [24]. This interaction involves responding immediately to various events or signals emitted by the plant. Consequently, the enforcer must address erroneous executions in CPS without delay and ensure continuous operation.

Different methods have been proposed to synthesize enforcers for CPSs. Authors in [6] first introduced a framework to synthesize enforcers for reactive systems, focusing solely on safety properties and considering untimed properties expressed as automata. Subsequent studies, including [21, 24, 25], extended this framework to include bi-directional runtime enforcement for CPSs. However, all these methods assume a system model in discrete time. Although

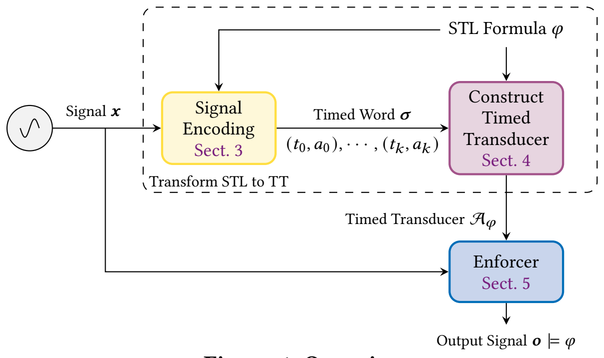

STL Formula φ
Signal $\boldsymbol{x}$
Signal Encoding Sect. 3
Timed Word $\boldsymbol{\sigma}$
$(t_0, a_0), \cdots, (t_k, a_k)$
Construct Timed Transducer Sect. 4
Transform STL to TT
Timed Transducer $\mathcal{A}_\varphi$
Enforcer Sect. 5
Output Signal $\boldsymbol{o} \models \varphi$

Figure 1: Overview

this assumption simplifies the modeling, it is devoid of the expressive power of continuous time specifications.

In this paper, we take a step further by proposing a uniform hierarchy to synthesize enforcers for CPSs operating in dense time, with properties expressed using STL. The enforcer processes an input signal $\boldsymbol{x}$ observed up to the current time t and generates an output signal $\boldsymbol{o}$ that satisfies the specified STL formula φ. As illustrated in Fig. 1, the enforcer encompasses three steps: (i) Encoding a signal $\boldsymbol{x}$ as a timed word in accordance with the STL formula. The timed word can be recognized by a timed automaton. This procedure is depicted as the yellow block in Fig. 1. (ii) Constructing a variant of Timed Automaton (TA) from the given STL formula. The TA, with both input and output, is represented as *Timed Transducer* (TT). The output of the TT indicates the enforcement strategy to be applied to the current input event (illustrated as the purple block in Fig. 1). (iii) Enforcing a signal $\boldsymbol{x}$ using the TT $\mathcal{A}_\varphi$ constructed from STL formula φ (represented as the blue block in Fig. 1). A significant advantage of our enforcer is that it *does not* require reachability analysis because the enforcement strategies used are encoded explicitly within the timed transducer. Our experimental results demonstrate the effectiveness of our approach and provides empirical evidence of its suitability for CPS.

Our enforcer is characterized by *soundness* (the output signal $\boldsymbol{o}$ satisfy φ), *transparency* (the input signal $\boldsymbol{x}$ are only modified when necessary), and *minimal modification* (the difference between the output and input signal values is minimal). Our work makes the following contributions:

(1) We propose a method to encode dense time signals into time words while preserving the information required to adjust the compliance of the signals with STL formulas (Sect. 3),
(2) we introduce a uniform approach to construct timed transducers against STL formulae, enabling these transducers to enforce the compliance of the STL formula on the input timed word (Sect. 4),
(3) we develop a method to minimally modify the signal to ensure its satisfaction against the given STL formula (Sect. 5),
(4) we provide experimental evidence to demonstrate the effectiveness of our approach (Sect. 6).

Organization. Sect. 2 provides a recap of important preliminaries and formally defines the problem. The process of encoding a signal into a timed word is detailed in Sect. 3. The transformation of STL into TTs is discussed in Sect. 4. Sect. 5 describes the method for signal modification and the comprehensive runtime enforcement algorithm for STL formulas. A relevant case study is presented in Sect. 6. Related works are reviewed in Sect. 7. Finally, the conclusions and future work are outlined in Sect. 8.

2 BACKGROUND AND PROBLEM FORMULATION

Let $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{Q}_{\geq 0}$ denote the set of natural numbers, real numbers, non-negative real numbers and non-negative rational numbers, respectively. For a set $A \subseteq \mathbb{R}$ and a real number $a \in \mathbb{R}$, the expression $a \oplus A$ is used to denote the set obtained by adding a to each element in A.

2.1 Timed Transducer

A Timed Transducer is a specialized version of a timed automaton [1] that is capable of both taking input and producing output. We provide the essential preliminaries of timed transducers below.

Timed Language. Let Σ denote a finite alphabet. A pair $(t, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ is called an *event*. A *timed word* over Σ is a finite sequence $\boldsymbol{\sigma} = (t_0, a_0)(t_1, a_1) \cdots (t_n, a_n) \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, where t_i is the *timestamp* indicating the global time at which the *action* a_i occurs, for all $0 \leq i \leq n$. A *timed language* $\mathcal{L}$ is a set of timed word, i.e., $\mathcal{L} \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^*$.

Timed Transducer. Let C be the set of clock variables. A *clock constraint* g is a Boolean combination of atomic constraints of the form $c \bowtie r$, with $c \in C$, $r \in \mathbb{Q}_{\geq 0}$, and $\bowtie \in \{\leq, <, \geq, >, =\}$. We use $\mathcal{G}(C)$ to denote the set of clock constraints. A *clock valuation* $v : C \mapsto \mathbb{R}_{\geq 0}$ is a function assigning a non-negative real value to each clock $c \in C$. We write $v \models g$ if the clock valuation v satisfies the clock constraints g. For $d \in \mathbb{R}_{\geq 0}$, let $v + d$ denote the clock valuation which maps every clock $c \in C$ to the value $v(c) + d$, and for a set $C' \subseteq C$, let $v[C' \leftarrow 0]$ denote the clock valuation which resets all clock variables in C' to 0 and agrees with v for other clocks in $C \setminus C'$. A timed transducer is defined as below:

Definition 1 (Timed transducer). A timed transducer is a tuple $\mathcal{A} = (L, l_0, C, \Sigma, \Lambda, \Delta, \lambda, F)$, where

- L is a finite set of locations;
- l_0 is the initial location;
- C is the set of clocks;
- Σ is the input alphabet;
- Λ is the output alphabet;
- $\Delta \subseteq L \times \Sigma \times \mathcal{G}(C) \times 2^C \times L$ is a finite set of transitions;
- $\lambda : \Delta \mapsto \Lambda$ is the output function that associates each transition with an output;
- $F \in L$ is a set of accepting locations;

A transition $\delta = (l, a, g, C', l')$ in Δ represents a jump from location l to l' by performing an action $a \in \Sigma$ when the constraint $g \in \mathcal{G}(C)$ is satisfied by the current clock valuation. The set C' indicates which clocks should be reset upon reaching l'.

A *state* q of $\mathcal{A}$ is a pair (l, v), where $l \in L$ denotes the location, and v is a clock valuation. A *run* ρ of $\mathcal{A}$ over an input timed word $\boldsymbol{\sigma} = (t_0, a_0)(t_1, a_1) \cdots (t_n, a_n)$ is a sequence $(l_0, v_0) \xrightarrow[b_0]{\tau_0, a_0} (l_1, v_1) \xrightarrow[b_1]{\tau_1, a_1} \cdots \xrightarrow[b_n]{\tau_n, a_n} (l_{n+1}, v_{n+1})$, where $\tau_i = t_i - t_{i-1}$ for $i = 1, 2, \ldots, n$ and $\tau_0 = t_0$, satisfying the following conditions:

(1) l_0 is the initial location and $v_0(c) = 0$ for all $c \in C$,
(2) For each $i = 0, 1, \cdots, n$, there is a transition $\delta_i = (l_i, a_i, g_i, C_i, l_{i+1}) \in \Delta$ such that $v_i + \tau_i \models g_i$ and $v_{i+1} = (v_i + \tau_i)[C_i \leftarrow 0]$,
(3) $\lambda(\delta_i) = b_i \in \Lambda$ for all $i = 0, 1, \cdots, n$.

The run ρ is *accepted* by $\mathcal{A}$ if $l_{n+1} \in F$. The output timed word induced by $\mathcal{A}$ is $\boldsymbol{\omega} = (t_0, b_0)(t_1, b_1) \cdots (t_n, b_n)$, sharing the same timestamp t_i $(i = 0, 1, \cdots, n)$ as the input timed word. We use the notation $[\![\mathcal{A}]\!](\boldsymbol{\sigma}) = \boldsymbol{\omega}$ to denote that $\mathcal{A}$ executes an accepted run over input timed word $\boldsymbol{\sigma}$ that induces output timed word $\boldsymbol{\omega}$.

Example 1. The $\mathcal{A}_P$ illustrated in Fig. 2 represents a TT for the property P: "*There should be a delay of at least* 5 *time units between any two read file requests*". This TT consists of locations $L = \{l_0, l_1, l_2\}$, with l_0 as the initial location and $\{l_0, l_1\}$ as the accepting locations, indicated by double circles. The input alphabet is $\Sigma = \{r, w\}$ with r for read requests and w for write requests. The output alphabet is $\{\top, \bot\}$ (denoted by green in Fig. 2), where $\top$ indicates a proper input that may lead to an accepted run, and $\bot$ indicates an improper input that leads to an unacceptable run. The transducer operates with one clock c.

Given the input timed word $\boldsymbol{\sigma} = (1, r)(4, w)(6, r)$, the run ρ of $\mathcal{A}$ progresses as follows:

$$\rho = (l_0, 0) \xrightarrow[\top]{1,r} (l_1, 0) \xrightarrow[\top]{3,w} (l_1, 3) \xrightarrow[\top]{2,r} (l_1, 0).$$

The output timed word of $\mathcal{A}_p$ over input timed word $\boldsymbol{\sigma}$ is $[\![\mathcal{A}_P]\!](\boldsymbol{\sigma}) = (1, \top)(4, \top)(6, \top)$, which indicates whether the transition at current timestamp results in an acceptable run. ◁

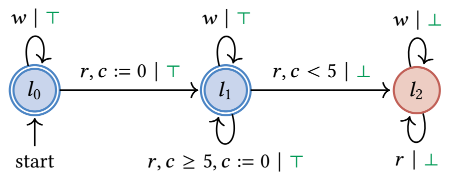


Figure 2: Timed Transducer $\mathcal{A}_P$

2.2 Signal Temporal Logic

Signal Temporal Logic [19] is a predicate logic used to describe and analyze continuous real-valued signals. Consider a signal $\boldsymbol{x} : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}^n$. For each predicate $p(\boldsymbol{x})$, there is a corresponding function $\mu_p : \mathbb{R}^n \mapsto \mathbb{R}$. The truth value of $p(\boldsymbol{x})$ at time t is defined as follows:

$$p(\boldsymbol{x}(t)) ::= \begin{cases} \top, & \text{if} \quad \mu_p(\boldsymbol{x}(t)) \geq 0, \\ \bot, & \text{if} \quad \mu_p(\boldsymbol{x}(t)) < 0. \end{cases}$$

We will use the notion $p(\boldsymbol{x}(t)) \equiv \mu_p(\boldsymbol{x}(t)) \geq 0$ to define a predicate $p(\boldsymbol{x})$ in this paper.

As demonstrated in [7], any STL formula can be equivalently converted into Negation Normal Form (NNF), in which negations appear only adjacent to predicates. In this paper, we considered *non-nested* STL formula in NNF, which can be defined recursively as below:

$$\phi ::= \top \mid p(\boldsymbol{x}) \mid \neg p(\boldsymbol{x}) \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2,$$
$$\varphi ::= \phi_1 \mathcal{U},I = 1_{[I]} \phi_2 \mid \phi_1 \mathcal{R},I = 1_{[I]} \phi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2\,,$$

where $\mathcal{U}_I$ and $\mathcal{R}_I$ are the *until* and *release* operators, respectively. $I = [t_1, t_2]$ is a *bounded* interval with *rational* endpoints (i.e., $t_1, t_2 \in \mathbb{Q}_{\geq 0}$)[1]. Note that $\mathcal{R}_I$ are the dual of $\mathcal{U}_I$, in a way that $\phi_1 \mathcal{R}_I \phi_2 \equiv \neg(\neg\phi_1 \mathcal{U}_I \neg\phi_2)$. We use $pd(\varphi)$ to denote the set of *predicates* in an STL formula φ. The NNF replaces the negation of a formula by including all operators and their duals in the grammar. Other temporal operators can be defined as syntactic sugars, e.g., $\Diamond_I \phi \equiv \top \mathcal{U}_I \phi, \Box_I \phi \equiv \bot \mathcal{R}_I \phi$.

Remark 1. *We employ NNF because it facilitates the subsequent transformation of an STL formula into a timed transducer. This choice is strategically important since timed transducer, acting as a special form of timed automata, are not closed under complementation. Thus, negating an STL formula would necessitate taking the complement of its corresponding timed automaton, which is problematic due to this lack of closure.*

The semantics of STL is defined as the satisfaction of a formula φ with respect to a signal $\boldsymbol{x}$ and time $t \in \mathbb{R}_{\geq 0}$.

Definition 2 (STL Semantics). The satisfaction of an STL formula φ at a given time t over a signal $\boldsymbol{x}$, denoted by $(\boldsymbol{x}, t) \models \varphi$, is inductively defined as follows:

$$\begin{array}{lll}
(\boldsymbol{x}, t) \models \top & & \\
(\boldsymbol{x}, t) \models p(\boldsymbol{x}) & \text{iff} & \mu_p(\boldsymbol{x}(t)) \geq 0 \\
(\boldsymbol{x}, t) \models \neg p(\boldsymbol{x}) & \text{iff} & \mu_p(\boldsymbol{x}(t)) < 0 \\
(\boldsymbol{x}, t) \models \varphi_1 \wedge \varphi_2 & \text{iff} & (\boldsymbol{x}, t) \models \varphi_1 \text{ and } (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (\boldsymbol{x}, t) \models \varphi_1 \text{ or } (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) \models \varphi_1 \mathcal{U}_I \varphi_2 & \text{iff} & \exists t' \in t \oplus I,\ (\boldsymbol{x}, t') \models \varphi_2 \\
& & \quad \text{and } \forall t'' \in [t, t'], (\boldsymbol{x}, t'') \models \varphi_1 \\
(\boldsymbol{x}, t) \models \varphi_1 \mathcal{R}_I \varphi_2 & \text{iff} & \forall t' \in t \oplus I,\ (\boldsymbol{x}, t') \models \varphi_2 \\
& & \quad \text{or } \exists t'' \in [t, t'], (\boldsymbol{x}, t'') \models \varphi_1
\end{array}$$

Intuitively, the subscript I in the until operator $\mathcal{U}_I$ defines the timing constraints under which a signal must *eventually* satisfy φ_2, while ensuring that φ_1 is satisfied beforehand. Similarly, the subscript I in the release operator $\mathcal{R}_I$ specifies the timing constraints in which a signal must *always* satisfy φ_2, unless φ_1 has been satisfied earlier. We say $\boldsymbol{x} \models \varphi$ if $(\boldsymbol{x}, 0) \models \varphi$.

Example 2. The following examples illustrate some properties defined by STL.

(1) $(\boldsymbol{x} \leq 30)\mathcal{U}_{[5,10]}(\boldsymbol{x} = 0)$: The value of the signal will be 0 at a time instant between 5 to 10 seconds; until then the value of the signal is less than 30.
(2) $\Box_{[0,\infty)}(\boldsymbol{x} < 3.5)$: The signal is always below 3.5.
(3) $\Diamond_{[0,30]}(\boldsymbol{x} > 100)$: At some time in the first 30 seconds, the value of the signal will exceed 100. ◁

2.3 Runtime Enforcement

The purpose of RE is to monitor input sequences produced by a running system and transform them into output sequences that adhere to a specified property φ. This is achieved using an enforcer.

[1]The endpoints of I are restricted to $\mathbb{Q}_{\geq 0}$ to facilitate encoding this into the clock constraints of the TT defined in Sect. 2.1

Constraints on an Enforcer. Let X denote the set of signals $\boldsymbol{x} : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}^n$. Some constraints are required on how enforcer E_φ for φ transforms a signal $\boldsymbol{x}$ at time t, to ensure that it performs correctly and minimally disruptively.

Definition 3 (Constraints on an Enforcer). Given an STL formula φ, an enforcer is a function $E_\varphi : X \mapsto X$ that satisfies the following conditions:

- *Soundness*:
$$\forall \boldsymbol{x} \in X,\ E_\varphi(\boldsymbol{x}) \models \varphi,$$
- *Transparency*:
$$\forall \boldsymbol{x} \in X,\ \boldsymbol{x} \models \varphi \implies E_\varphi(\boldsymbol{x}) = \boldsymbol{x},$$
- *Minimal Modification*:
$$\forall \boldsymbol{x} \in X,\ \boldsymbol{x} \not\models \varphi \implies E_\varphi(\boldsymbol{x}) = \arg\min_{\boldsymbol{o} \in O} ||\boldsymbol{x} - \boldsymbol{o}||_s,$$
where $O = \{\boldsymbol{o} \mid \boldsymbol{o} \models \varphi \wedge |\boldsymbol{x}| = |\boldsymbol{o}|\}$, $|\boldsymbol{x}|$ is the length of a signal, and $||\boldsymbol{x} - \boldsymbol{o}||_s ::= \max_t ||\boldsymbol{x}(t) - \boldsymbol{o}(t)||$ is the distance between signals, with $|| \cdot ||$ being the Euclidean norm in $\mathbb{R}^n$.

Intuitively, *soundness* ensures that the output signal complies with the specified STL formula φ. *Transparency* stipulates that if the input signal $\boldsymbol{x}$ already meets φ, the enforcer should not alter it, but rather transmit the original signal $\boldsymbol{x}$ as the output. *Minimal modification* requires that if the input signal $\boldsymbol{x}$ does not satisfy φ, the enforcer should adjust it to ensure compliance with φ, while keeping the modifications as minimal as possible relative to the original signal $\boldsymbol{x}$.

Problem Formulation. With all the preliminary details established, we now formally define the problem under consideration in this paper as follows:

Synthesis of Enforcer. Given an STL formula φ, construct an enforcer $E_\varphi : X \mapsto X$ for φ that satisfies the *soundness*, *transparency*, and *minimal modification* conditions as per Def. 3.

3 SIGNAL ENCODING

In this section, we will introduce the procedure for encoding a signal into a timed word. This step is essential because we aim to enforce a signal using a TT, but a signal, defined as a real-valued function over dense time, is not directly compatible with TT.

The encoding process, applied to a given signal $\boldsymbol{x}$ with respect to an STL formula φ, involves recording the truth value of predicates of φ at both *variable points* and *relevant points* within the signal. We will now provide a detailed explanation of this encoding procedure.

Variable Points. Intuitively, a variable point is where the truth value of a predicate regarding the signal changes. The concept of variable points is as below.

Definition 4 (Variable Point [3, Def. 2.8]). Given a signal $\boldsymbol{x} : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}^n$, a time point $\tau \in \mathbb{R}_{\geq 0}$ is a variable point of $\boldsymbol{x}$ with respect to a predicate $p(\boldsymbol{x})$ if for some neighborhood B containing τ, there are different truth values u and v such that $p(\boldsymbol{x}) = u$ for every $t \in B \cap [0, \tau)$ and $p(\boldsymbol{x}) = v$ for every $t \in B \cap (\tau, +\infty)$.

In this paper, we limit our focus to *non-Zeno* signals constrained within a *bounded* time frame. Consequently, such signals possess a finite number of variable points. There is a notable characteristic of variable points, formalized in the following proposition:

Proposition 1. *Given a signal $\boldsymbol{x}$ and a predicate $p(\boldsymbol{x})$, assume $t_0 < t_1 < \cdots < t_k$ denote all the variable points of $\boldsymbol{x}$ with respect to $p(\boldsymbol{x})$. It is then established that the truth value of $p(\boldsymbol{x})$ remains constant within each open interval (t_i, t_{i+1}) for all indices $i = 0, 1, \ldots, k$, with t_{k+1} being the endpoint of the signal $\boldsymbol{x}$.*

Consequently, the signal $\boldsymbol{x}$ can be effectively encoded as a timed word:
$$\boldsymbol{\sigma} = (t_0, a_0), (t_1, a_1), \cdots, (t_k, a_k),$$
where each t_i denotes a variable point with respect to $p(\boldsymbol{x})$, and each a_i represents the truth value of $p(\boldsymbol{x})$ within the interval (t_i, t_{i+1}) for indices $i = 0, 1, \ldots, k$. This encoding method ensures that $\boldsymbol{\sigma}$ comprehensively captures all information about $\boldsymbol{x}$ with respect to the predicate $p(\boldsymbol{x})$.

The set of variable points of a signal with respect to an STL formula φ is defined as the union of the sets of variable points for all predicates $p \in pd(\varphi)$[2]. Accordingly, the timed word can be constructed based on these combined variable points. The following example provides a comprehensive illustration of this process:

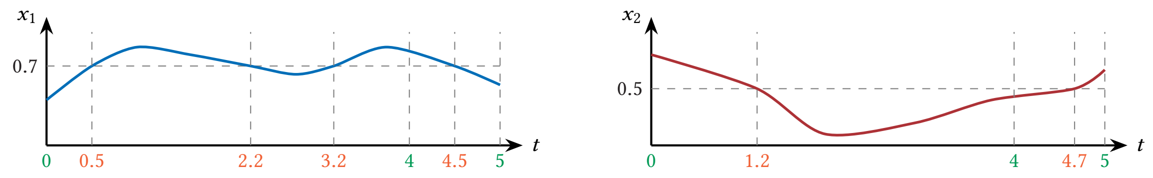
Example 3. Consider the signal $\boldsymbol{x} = (x_1, x_2)$ illustrated in Fig. 3, and the STL formula $\varphi = p_1 \mathcal{U}_{[4,5]} p_2$, with $p_1 \equiv x_1 - 0.7 \geq 0$, and $p_2 \equiv x_2 - 0.5 \geq 0$. The time word for $\boldsymbol{x}$ with respect to φ is given by:

$$(0.5, p_1 \wedge p_2), (1.2, p_1 \wedge \neg p_2), (2.2, \neg p_1 \wedge \neg p_2), (3.2, p_1 \wedge \neg p_2),$$
$$(4.5, \neg p_1 \wedge \neg p_2), (4.7, \neg p_1 \wedge p_2) \quad \triangleleft$$

The values of a signal $\boldsymbol{x}$ that may lead to variable points against the predicate $p(\boldsymbol{x})$ can be determined by solving the equation $\mu_p(x) = 0$. Depending on the structure of $\mu_p(x)$, different methods can be used to find or approximate the root of $\mu_p(x)$: (i) Gaussian elimination is suitable for linear forms of $\mu_p(x)$, (ii) Newton-Raphsom method can be used for polynomial forms of $\mu_p(x)$, (iii) For more complex forms, such as transcendental equations, the Real roots isolation method [12] can be used to approximate the intervals of the roots. Using $vv(p)$ to denote the set of such valuations against predicate p, then $vv(\varphi) = \cup_{p \in pd(\varphi)} vv(p)$ for a given STL formula φ.

Variable points are effective for documenting changes in a signal in accordance with a specific STL formula. However, they may not provide sufficient information for *enforcing* compliance with an STL property. Consider, for example, the scenario depicted in Exmp. 3, where the proposition p_1 is false within the time interval $[0, 0.5)$. This condition leads to the STL formula φ not being satisfied by the signal $\boldsymbol{x}$. However, in the absence of an input event before $t = 0.5$ — specifically, an input at $t = 0$ — no transducer can confirm that φ is unsatisfied by the signal, nor can it enforce the signal accordingly. To address this, it becomes necessary to incorporate *relevant points* tailored to an STL formula, ensuring all relevant events are recorded.

[2]Recall that $pd(\varphi)$ is defined as the set of predicates in an STL formula φ in Sect. 2



$$\begin{array}{r}\text{timestamps}\\ \text{actions}\end{array}\begin{pmatrix}0\\ \neg p_1\wedge p_2\end{pmatrix}\begin{pmatrix}0.5\\ p_1\wedge p_2\end{pmatrix}\begin{pmatrix}1.2\\ p_1\wedge\neg p_2\end{pmatrix}\begin{pmatrix}2.2\\ \neg p_1\wedge\neg p_2\end{pmatrix}\begin{pmatrix}3.2\\ p_1\wedge\neg p_2\end{pmatrix}\begin{pmatrix}4\\ p_1\wedge\neg p_2\end{pmatrix}\begin{pmatrix}4.5\\ \neg p_1\wedge\neg p_2\end{pmatrix}\begin{pmatrix}4.7\\ \neg p_1\wedge p_2\end{pmatrix}\begin{pmatrix}5\\ \neg p_1\wedge p_2\end{pmatrix}$$

Figure 3: Signal Encoding against Formula $p_1\mathcal{U}_{[4,5]}p_2$

Relevant Points. Intuitively, relevant points include the time points that correspond to the interval boundaries of the given formula and the initial instant of the given signal. These points mark where the satisfaction requirements for predicates may change. For example, in the formula $p_1\mathcal{U}_{[t_1,t_2]}p_2$, before t_1, ensuring that p_1 is satisfied suffices. However, after t_1, it is also necessary to verify the satisfaction of p_2.

We proceed to inductively define relevant points of an STL formula as follows:

Definition 5 (Relevant Point). Given an STL formula φ, the set of relevant points $rp(\varphi)$ is inductively defined by:

$$rp(\top)=\emptyset,\quad rp(\varphi_1\wedge\varphi_2)=rp(\varphi_1)\cup rp(\varphi_2),$$
$$rp(p(x))=\{0\},\quad rp(\varphi_1\vee\varphi_2)=rp(\varphi_1)\cup rp(\varphi_2),$$
$$rp(\varphi_1\mathcal{U}_{[t_1,t_2]}\varphi_2)=\{t_1,t_2\}\cup rp(\varphi_1)\cup rp(\varphi_2),$$
$$rp(\varphi_1\mathcal{R}_{[t_1,t_2]}\varphi_2)=\{t_1,t_2\}\cup rp(\varphi_1)\cup rp(\varphi_2).$$

The actions of events at time points $t\in rp(\varphi)$ reflect the truth values of all predicates in φ at time t, which can be directly sampled in real-time from the signal. Consider the following example for further illustration:

Example 4. Continuing Exmp. 3, recall that $\varphi=p_1\mathcal{U}_{[4,5]}p_2$. The set of relevant point $rp(\varphi)=\{0,4,5\}$, and the events at relevant points with respect to signal $\boldsymbol{x}=(x_1,x_2)$ is:

$$(0,\neg p_1\wedge p_2),(4,p_1\wedge\neg p_2),(5,\neg p_1\wedge p_2).$$

Consequently, the complete time word encoded from $\boldsymbol{x}$ with respect to the STL formula φ is depicted in Fig. 3. The events at variable points are highlighted in orange, while the events at relevant points are marked in green. ◁

Signal encoding. We now give the process of signal encoding. The signal encoding process, as outlined in Alg. 1, operates in real-time by monitoring changes in the truth values of predicates within the given STL formula. In an infinite loop, the algorithm waits for the input signal (Line 3) at current time t (the function current_time() can be used to get the current time as shown in Line 4). It updates the truth values CurrPred of all predicates (Line 5) with respect to the current signal values. An event is emitted (Line 7) whenever a variable point or relevant point is met (Line 6).

Remark 2. *During the signal encoding process, we can proactively identify variable points and relevant points, allowing for the enforcement of the signal before any actual violations occur. This proactive*

Algorithm 1: SignEncode(φ)

```
  Require: $\varphi$: STL formula
  Ensure : $\sigma$: time word encoded from $\boldsymbol{x}$
1 Rele ← $rp(\varphi)$, Vari ← $vv(\varphi)$, Pred ← $pd(\varphi)$;
2 while true do
3     $\boldsymbol{x}$ ← await_signal();
4     $t$ ← current_time() ;     // Get the current time $t$
5     CurrPred ← Truth values of predicates $p \in$ Pred with
        respect to $\boldsymbol{x}$ at $t$;
6     if $\boldsymbol{x}(t) \in$ Vari or $t \in$ Rele then
7         Emit ($t$, CurrPred);
```

enforcement can be achieved by: (i) expanding the values at a variable point into its surrounding neighborhood, and (ii) slightly adjusting the timing of relevant points in $rp(\varphi)$ to check these points in advance.

4 CONSTRUCTING TIMED TRANSDUCER FROM STL

In this section, we outline a methodology for constructing a TT based on an STL. This TT processes the encoded timed word discussed in Sect. 3 and generates output for the corresponding enforcement strategy.

The construction process is inspired by the compositional hierarchy utilized in [11] for building a TT for metric interval temporal logic (MITL)[2]. We have adopted this methodology and enhanced its applicability: (i) It is suitable for STL, accommodating temporal operators with punctual intervals (e.g., $\mathcal{U}_{[t_1,t_1]}$), (ii) It is appropriate for runtime enforcement. The output of the TT serves as an enforcement strategy, which pinpoints the specific predicate causing the STL formula violation. This identification allows us to precisely modify only the signals involved in the failure, rather than altering all signals indiscriminately.

Initially, we will describe the construction of the TT for the temporal operators $\mathcal{U}_I$ and $\mathcal{R}_I$ used in the normal form of Sect. 2.2. Subsequently, we will present the method for composing these operators according to the structure of the STL formula.

4.1 Timed Transducer for $\mathcal{U}_I$ and $\mathcal{R}_I$

TT for $p_1\mathcal{U}_{[t_1,t_2]}p_2$. Transducer $\mathcal{A}_{\mathcal{U}}$ is designed to enforce a signal according to the STL formula $p_1\mathcal{U}_{[t_1,t_2]}p_2$. The structure of $\mathcal{A}_{\mathcal{U}}$ is defined as follows:

- $L=\{l_0,l_1,l_2,l_3\}$;

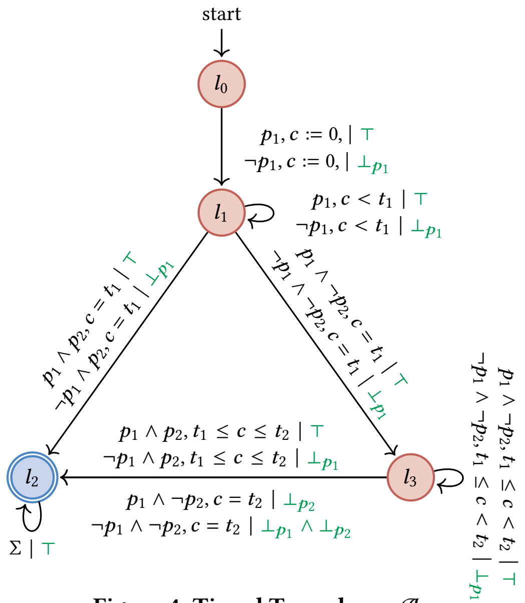

start
l_0
l_1
l_2
l_3
$p_1, c := 0, \mid \top$
$\neg p_1, c := 0, \mid \bot_{p_1}$
$p_1, c < t_1 \mid \top$
$\neg p_1, c < t_1 \mid \bot_{p_1}$
$p_1 \wedge p_2, c = t_1 \mid \top$
$\neg p_1 \wedge p_2, c = t_1 \mid \bot_{p_1}$
$p_1 \wedge \neg p_2, c = t_1 \mid \top$
$\neg p_1 \wedge \neg p_2, c = t_1 \mid \bot_{p_1}$
$p_1 \wedge p_2, t_1 \leq c \leq t_2 \mid \top$
$\neg p_1 \wedge p_2, t_1 \leq c \leq t_2 \mid \bot_{p_1}$
$p_1 \wedge \neg p_2, c = t_2 \mid \bot_{p_2}$
$\neg p_1 \wedge \neg p_2, c = t_2 \mid \bot_{p_1} \wedge \bot_{p_2}$
$p_1 \wedge \neg p_2, t_1 \leq c < t_2 \mid \top$
$\neg p_1 \wedge \neg p_2, t_1 \leq c < t_2 \mid \bot_{p_1}$
$\Sigma \mid \top$

Figure 4: Timed Transducer $\mathcal{A}_{\mathcal{U}}$

- $l_0 = l_0$;
- $C = \{c\}$;
- $\Sigma = \{p_1, p_2\}$;
- $\Lambda = \{\top, \bot_{p_1}, \bot_{p_2}\}$;
- $F = \{l_2\}$.

The set of transitions Δ and the corresponding outputs λ of $\mathcal{A}_{\mathcal{U}}$ is depicted in Fig. 4.

In the output alphabet, $\top$ represents the strategy of making no change to the signal value, while $\bot_{p_i}$ indicates that the signal value should be modified to satisfy the predicate p_i. Essentially, an output of $\bot_{p_i}$ suggests how the input action should be modified to achieve a $\top$ output. For instance, if the transition from l_1 to l_2 in Fig. 4 has an input action of $\neg p_1 \wedge p_2$ and outputs $\bot_{p_1}$, it implies that the input should be changed to $p_1 \wedge p_2$ to ensure a $\top$ output in this transition.

We propose below the equivalence of *Until* operator of STL and its transducer.

Proposition 2. *Let $\boldsymbol{x}$ be a signal and $\boldsymbol{\sigma}$ denote its encoded timed word against the STL formula $p_1\mathcal{U}_{[t_1,t_2]}p_2$. Define $\boldsymbol{\omega}_\top$ as the timed word where all event actions are $\top$. The following equivalence is then established:*

$$[\![\mathcal{A}_{\mathcal{U}}]\!](\boldsymbol{\sigma}) = \boldsymbol{\omega}_\top \iff \boldsymbol{x} \models p_1\mathcal{U}_{[t_1,t_2]}p_2$$

The proof relies on the case analysis based on the events received in time intervals $[0, t_1]$ and $(t_1, t_2]$. The detail proof can be found in the extended version [32].

TT for $p_1\mathcal{R}_{[t_1,t_2]}p_2$. Here, we detail the construction of the TT $\mathcal{A}_{\mathcal{R}}$. The structure of the TT $\mathcal{A}_{\mathcal{R}}$ is defined as follows:

- $L = \{l_0, l_1, l_2, l_3\}$;
- $l_0 = l_0$;
- $C = \{c\}$;
- $\Sigma = \{p_1, p_2\}$;
- $\Lambda = \{\top, \bot_{p_1}, \bot_{p_2}\}$;

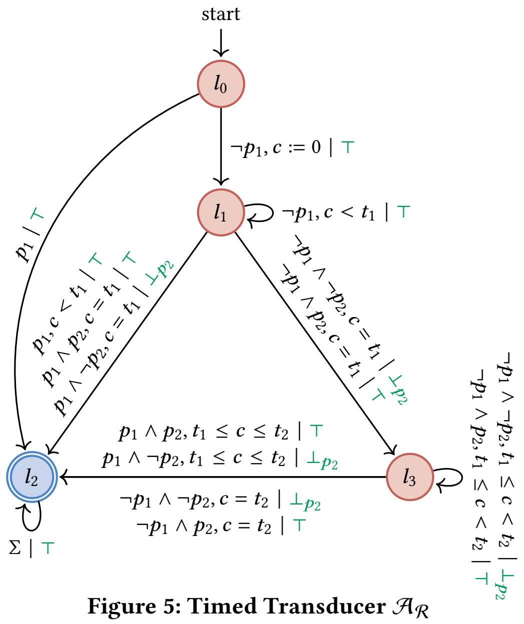

start
l_0
l_1
l_2
l_3
$\neg p_1, c := 0 \mid \top$
$p_1 \mid \top$
$\neg p_1, c < t_1 \mid \top$
$p_1, c < t_1 \mid \top$
$p_1 \wedge p_2, c = t_1 \mid \top$
$p_1 \wedge \neg p_2, c = t_1 \mid \bot_{p_2}$
$\neg p_1 \wedge \neg p_2, c = t_1 \mid \bot_{p_2}$
$\neg p_1 \wedge p_2, c = t_1 \mid \top$
$p_1 \wedge p_2, t_1 \leq c \leq t_2 \mid \top$
$p_1 \wedge \neg p_2, t_1 \leq c \leq t_2 \mid \bot_{p_2}$
$\neg p_1 \wedge \neg p_2, c = t_2 \mid \bot_{p_2}$
$\neg p_1 \wedge p_2, c = t_2 \mid \top$
$\neg p_1 \wedge \neg p_2, t_1 \leq c < t_2 \mid \bot_{p_2}$
$\neg p_1 \wedge p_2, t_1 \leq c < t_2 \mid \top$
$\Sigma \mid \top$

Figure 5: Timed Transducer $\mathcal{A}_{\mathcal{R}}$

- $F = \{l_2\}$.

where Δ and the corresponding λ are given in Fig. 5.

We propose below the equivalence of *Release* operator of STL and its transducer.

Proposition 3. *Let $\boldsymbol{x}$ be a signal and $\boldsymbol{\sigma}$ denote its encoded timed word against the given STL formula $p_1\mathcal{R}_{[t_1,t_2]}p_2$. Let $\boldsymbol{\omega}_\top$ be defined as before. The following equivalence is then established:*

$$[\![\mathcal{A}_{\mathcal{R}}]\!](\boldsymbol{\sigma}) = \boldsymbol{\omega}_\top \iff \boldsymbol{x} \models p_1\mathcal{R}_{[t_1,t_2]}p_2$$

Remark 3. *Essentially, the TT we constructed is* self-correcting*; that is, any transition within the TT has the potential to result in an acceptable run. This allows us to use such a TT to enforce a signal without worrying about the TT entering a violation state where no acceptable run exists.*

4.2 Compositionally Constructing the Entire Timed Transducer

In this paper, because we consider non-nested STL, the possible connections between two sub-formulas containing temporal operators (i.e., $p_1\mathcal{U}_I p_2$ or $p_1\mathcal{R}_I p_2$) are limited to either conjunction or disjunction. Consequently, it is sufficient to define the product of TTs we constructed in Sect. 4.1 according to $\wedge$ or $\vee$.

$\wedge$-Product. We will first explain how to construct TT of property in the form of $\varphi_1 \wedge \varphi_2$ by taking product between TTs, where the TTs for φ_1 and φ_2 have been constructed as $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Definition 6 ($\wedge$-Product). Given two TTs $\mathcal{A}_1 = (L_1, l_0^1, C_1, \Sigma_1, \Lambda_1, \Delta_1, \lambda_1, F_1)$ and $\mathcal{A}_2 = (L_2, l_0^2, C_2, \Sigma_2, \Lambda_2, \Delta_2, \lambda_2, F_2)$[3], the $\wedge$-product automaton $\mathcal{A}_1 \times_\wedge \mathcal{A}_2 ::= (L, l_0, C, \Sigma, \Lambda, \Delta, \lambda, F)$, where

- $L = L_1 \times L_2$,

[3]To avoid multiple subscripts, the indices of automata for the initial condition l_0 have been moved from superscript to subscript for both $\mathcal{A}_1$ and $\mathcal{A}_2$

- $l_0 = (l_0^1, l_0^2)$,
- $C = C_1 \cup C_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $\Lambda = \Lambda_1 \cup \Lambda_2$,
- $\delta = \big((l_1, l_2), (a_1, a_2), g_1 \wedge g_2, C_1' \cup C_2', (l_1', l_2')\big) \in \Delta$ iff $\delta_1 = (l_1, a_1, g_1, C_1', l_1') \in \Delta_1$ and $\delta_2 = (l_2, a_2, g_2, C_2', l_2') \in \Delta_2$,
- $\lambda(\delta) = \lambda_1(\delta_1) \wedge \lambda_2(\delta_2)$,
- $F = F_1 \times F_2$.

$\vee$-Product. We now explain how to construct TT of property in the form of $\varphi_1 \vee \varphi_2$ by taking product between TTs, where the TTs for φ_1 and φ_2 have been constructed as $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Definition 7 ($\vee$-product). Given two TTs $\mathcal{A}_1 = (L_1, l_0^1, C_1, \Sigma_1, \Lambda_1, \Delta_1, \lambda_1, F_1)$ and $\mathcal{A}_2 = (L_2, l_0^2, C_2, \Sigma_2, \Lambda_2, \Delta_2, \lambda_2, F_2)$, the $\vee$-product automaton $\mathcal{A}_1 \times_\vee \mathcal{A}_2 ::= (L, l_0, C, \Sigma, \Lambda, \Delta, \lambda, F)$, where

- $L = L_1 \times L_2$,
- $l_0 = (l_0^1, l_0^2)$,
- $C = C_1 \cup C_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $\Lambda = \Lambda_1 \cup \Lambda_2$,
- $\delta = \big((l_1, l_2), (a_1, a_2), g_1 \wedge g_2, C_1' \cup C_2', (l_1', l_2')\big) \in \Delta$ iff $\delta_1 = (l_1, a_1, g_1, C_1', l_1') \in \Delta_1$ and $\delta_2 = (l_2, a_2, g_2, C_2', l_2') \in \Delta_2$,
- $\lambda(\delta) = \lambda_1(\delta_1) \vee \lambda_2(\delta_2)$,
- $F = (F_1 \times L_2) \cup (L_1 \times F_2)$.

Essentially, the primary distinction between the $\wedge$-product and the $\vee$-product lies in the output function λ and the acceptance condition F. For a formula of the form $\varphi_1 \wedge \varphi_2$, the signal must satisfy both φ_1 and φ_2. Therefore, a transition in the product TT is considered 'good' (i.e., the output action is $\top$) iff the transitions in both $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ are 'good'. Additionally, the acceptance condition must ensure that the acceptance locations of both TTs are reached. Conversely, for a formula of the form $\varphi_1 \vee \varphi_2$, it is sufficient for the signal to satisfy either φ_1 or φ_2.

By induction on the structure of a given STL formula, the following proposition holds:

Proposition 4. *Let $\boldsymbol{x}$ be a signal and $\boldsymbol{\sigma}$ denote its encoded timed word against the given STL formula φ_1 op φ, where op $\in \{\wedge, \vee\}$. Let $\boldsymbol{\omega}_\top$ be defined as in Prop. 2. The following equivalence is then established:*

$$[\![\mathcal{A}_{\varphi_1} \times_{op} \mathcal{A}_{\varphi_2}]\!](\boldsymbol{\sigma}) = \boldsymbol{\omega}_\top \iff \boldsymbol{x} \models \varphi_1 \, op \, \varphi_2.$$

5 RUNTIME ENFORCEMENT USING TRANSDUCER

With all the preparatory work, we will present our runtime enforcement mechanism in this section. We will first describe an optimization-based method to *minimally modify* the signal according to the output of the TT as detailed in Sect. 4. Then, we will introduce the enforcer against an STL formula, which will be outlined through a designated algorithm.

Minimally modifying the signal. For a given STL formula φ, let $x = \boldsymbol{x}(t)$[4] be the value of the signal at the timestamp t. Assume the output action of $\mathcal{A}_\varphi$ induced by the input (t, a) is $\bot_{p_k}$. The signal can be modified by solving the following optimization problem:

$$\begin{aligned} \text{Minimize:} \quad & \|y - x^{p_k}\| \\ \text{Subject to:} \quad & \mu_i(x[x^{p_k}/y]) \geq 0, \quad \forall p_i \in a, \\ & \mu_j(x[x^{p_k}/y]) < 0, \quad \forall \neg p_j \in a, \\ & \mu_k(y) \bowtie 0, \end{aligned} \tag{1}$$

where $\bowtie$ is $\geq$ if $\neg p_k \in a$ and $\bowtie$ is $<$ otherwise. Here, x^{p_k} denotes the components of x related to the predicate p_k, y is the decision variable of the optimization problem, representing the modified value of the signal at t. Notation $x[x^{p_k}/y]$ denotes the vector obtained by replacing the occurrences of x^{p_k} in x with y. For an intuitive illustration, see Fig. 6, where $y = (y_1, y_2, y_3)$.

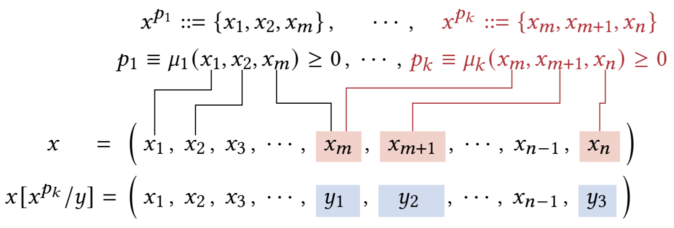

$x^{p_1} ::= \{x_1, x_2, x_m\}$, $\cdots$, $x^{p_k} ::= \{x_m, x_{m+1}, x_n\}$
$p_1 \equiv \mu_1(x_1, x_2, x_m) \geq 0$, $\cdots$, $p_k \equiv \mu_k(x_m, x_{m+1}, x_n) \geq 0$
$x = (x_1, x_2, x_3, \cdots, x_m, x_{m+1}, \cdots, x_{n-1}, x_n)$
$x[x^{p_k}/y] = (x_1, x_2, x_3, \cdots, y_1, y_2, \cdots, x_{n-1}, y_3)$

Figure 6: Illustration of x^{p_k} and $x[x^{p_k}/y]$.

We refer to this procedure as Modify (x, a, b, φ), where x represents the value of the signal, a is the input action of the TT, b is the corresponding output action, and φ is the STL formula. The following proposition confirms the robustness of our minimal modification method:

Proposition 5. *If the optimization problem in Eq. (1) is solvable, then this procedure maintains the minimal modification requirement as per Def. 3.*

Remark 4. *Eq. (1) can be solved using different methods, depending on the constraints provided in the predicate functions μ_p in the STL formula. If all of the μ_p are linear, then Eq. (1) can be solved using Quadratic Programming (QP). If the μ_p are polynomial, Eq. (1) can be transformed into Semidefinite Programming (SDP) by using Putinar's Positivstellensatz [26].*

Enforcer. We are now ready to present our runtime enforcement algorithm, as shown in Alg. 2. Assume we have a signal $\boldsymbol{x}$ to be enforced against an STL formula φ. The algorithm begins by computing its TT $\mathcal{A}_\varphi$ following the method described in Sect. 4 (Line 1 in Alg. 2). Subsequently, as the signal $\boldsymbol{x}$ is received, it is encoded into input events. The enforcer E_φ in Alg. 2 then traverses $\mathcal{A}_\varphi$ and generates the output events. Depending on this output, the signal is modified (if required) and released.

The algorithm proceeds as follows: `currState` monitors the current state of the TT, which includes the current location and clock valuation in the timed transducer. `currState` is initially set to the starting state of $\mathcal{A}_\varphi$ (Line 2). It then enters an infinite loop (Line 3) until an event is detected from Alg. 1 (Line 4).

Upon receiving an event, the transducer $\mathcal{A}_\varphi$ transitions, updates the `currState`, and gives the output b according to the transition (Line 5). If the output is anything other than $\top$, the transducer minimally modifies the signal before it is released (Line 7).

[4]Note that the bold $\boldsymbol{x}$ represents a signal, while the non-bold x is an n-dimensional real vector. We introduce the non-bold x here to ease the symbolic burden in Eq. (1)

state before transition	timestamp	input action	state after transition	output action
$(l_0, 0)$	0	$\neg p_1 \wedge p_2$	$(l_1, 0)$	$\bot_1$
$(l_1, 0)$	0.5	$p_1 \wedge p_2$	$(l_1, 0.5)$	$\top$
$(l_1, 0.5)$	1.2	$p_1 \wedge \neg p_2$	$(l_1, 1.2)$	$\top$
$(l_1, 1.2)$	2.2	$\neg p_1 \wedge \neg p_2$	$(l_1, 2.2)$	$\bot_1$
$(l_1, 2.2)$	3.2	$p_1 \wedge \neg p_2$	$(l_1, 3.2)$	$\top$
$(l_1, 3.2)$	4	$p_1 \wedge \neg p_2$	$(l_3, 4)$	$\top$
$(l_3, 4)$	4.5	$\neg p_1 \wedge \neg p_2$	$(l_3, 4.5)$	$\bot_1$
$(l_3, 4.5)$	4.7	$\neg p_1 \wedge p_2$	$(l_2, 4.7)$	$\bot_1$

Table 1: Transitions in TT of $p_1 \mathcal{U}_{[4,5]} p_2$

Algorithm 2: Algorithm Enforcer $E_\varphi(\boldsymbol{x})$

```
1 $\mathcal{A}_\varphi \leftarrow$ TT constructed from $\varphi$ ;
2 currState $\leftarrow [l_0, c := 0]$;
3 while true do
4     $(t, a) \leftarrow$ event emitted by Alg. 1;
5     currState, $b$ = make_transition$_{\mathcal{A}_\varphi}$(currState, $t$, $a$);
6     if $b \neq \top$ then
7         $\boldsymbol{x}(t)$ = Modify($\boldsymbol{x}(t), a, b, \varphi$);
8     release $\boldsymbol{x}$;
```

Remark 5. *For the outputs of TT in the form $\bot_{p_1} \vee \bot_{p_2}$, the enforcer will compute the optimization problem for both p_1 and p_2 independently and enforce the predicate that requires the minimal change, thereby adhering to the principle of minimal modification.*

The following example illustrates how our enforcer operates.

Example 5 (Enforcement of STL formula on a timed word). Continuing to Exmp. 4, recall that the STL property is defined as $p_1 \mathcal{U}_{[4,5]} p_2$, where $p_1 \equiv x_1 \geq 0.7$, $p_2 \equiv x_2 \geq 0.5$. Table 1 gives the steps of enforcement of the timed word using Until Transducer. The signal at time points $\{0, 2.2, 4.5, 4.7\}$ are modified to satisfy the STL formula. The modified signal is shown in Fig. 7. ◁

The following theorem states the correctness of the enforcer described in Alg. 2

Theorem 1. *Given an STL formula φ and a signal $\boldsymbol{x}$, the enforcer E_φ in Alg. 2 can enforce $\boldsymbol{x}$ to satisfy φ, while ensuring that the* soundness, transparency, *and* minimal modification *conditions in Def. 3 are met.*

Proof. The transparency of the enforcer is a direct result of Prop. 2, Prop. 3, and Prop. 4, as Alg. 2 will not modify the signal under the $\top$ output of the TT. The soundness of the enforcer is ensured by observing the truth that, the $\bot_p$ outputs of TT essentially indicate how to modify the input action to those inputs that can lead to a $\top$ output. The minimal modification condition is ensured by Prop. 5. □

Complexity Analysis. The time complexity of Alg. 2 is multifaceted. The time complexity of the function make_transition$_{\mathcal{A}_\varphi}$ is $O(m \times n)$, where m is the number of states in the TT and n is the size of the input alphabet. The time complexity of the function

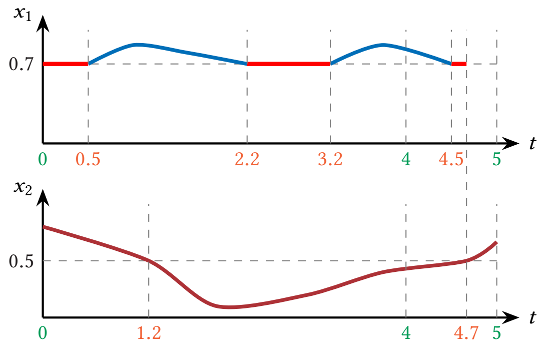


Figure 7: Enforced Signal in Exmp. 5

Modify depends on the structure of the predicate function in the STL formula; it will be polynomial in the number of decision variables when the predicate functions are linear [20].

Other procedures, such as constructing the TT from the STL formula in Sect. 4 (polynomial in the size of the TT, primarily influenced by the composition operator), and computing the values of the signal leading to the variable points in Sect. 3 (achieving quadratic convergence with the Newton-Raphson method), may be time-consuming. However, both procedures can be performed *offline*, thus they do not impact the efficiency of our runtime enforcement algorithm.

6 CASE STUDY

We developed a prototype of our runtime enforcement algorithm in Python and applied this prototype to case studies on Autonomous Vehicles (AVs) to demonstrate the efficiency and scalability of our method[5]. Three cases were considered in the experiments. The first case addresses the property of 'safe stopping of AVs', the second focuses on 'safe charging of AVs', while the third focuses on 'safe deceleration of AVs'. All the cases underscore the efficiency (Sect. 6.1) and scalability (Sect. 6.2) of our method.

6.1 Efficiency Evaluation

Safe stopping of AVs. Consider a scenario in which an AV is required to decelerate to a complete stop when approaching a red light or a designated stop point. This requirement is expressed by the property $(v \leq 30)\mathcal{U}_{[5,10]}(v = 0)$. This stipulates that *the speed of the vehicle must ultimately reach* 0 *within a time frame of* 5 *to* 10 *seconds, while maintaining a speed no larger than* 30 *until then.*

The results of our experiment are depicted in Fig. 8, where the blue signal represents the output after enforcement, while the orange one is the original signal. These results demonstrate that the enforcement monitor effectively adjusted the signal to ensure compliance with the STL property, while maintaining transparency and minimal modification. Specifically, the enforcer precisely addressed the four instances where the speed exceeded 30 (sudden speed spikes), applying only the necessary changes without superfluous adjustments to the signal.

Safe charging of AVs. Consider a scenario in the battery charging systems of AVs. Normally, the current stays within a safe range throughout a specified interval. If, however, the voltage reaches a

[5]Code available at https://github.com/Han-SU/stl-enforcement-experiments.git

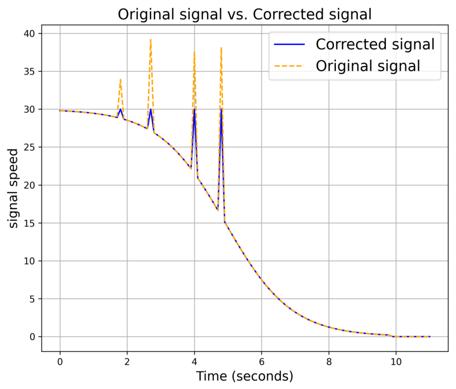


Figure 8: Enforcement of Speed Signal against Safe Stopping Property

specific volts, the system switches to a charging mode designed to safely handle higher currents. This condition is formally represented by the property $(V = 4.2)\mathcal{R}_{[2,10]}(I < 10)$, which indicates that *the current will not exceed* 10 *within a timeframe of* 2 *to* 10 *seconds, unless the voltage reaches* 4.2 *volts earlier.*

The results of our experiment are depicted in Fig. 9, where the blue signal represents the output after enforcement, and the orange signal is the original one. These results illustrate that during the interval from 2 to 10 seconds, the current I is minimally adjusted to remain below 10, provided that the voltage V does not reach 4.2 volts.

Safe deceleration of AVs. Consider a scenario of coordinated deceleration for stability and safety in AVs, where both wheels and motor controls slow down together, helping avoid sudden stops or imbalances. It is a dual-redundant safety feature: The wheel subsystem must ensure that its value does not exceed 30 within the timeframe and ultimately reaches zero between 5 and 10 seconds. Simultaneously, the motor control subsystem has the same requirement. This condition is formally represented by the property $(w \leq 30)\mathcal{U}_{[5,10]}(w = 0) \land (m \leq 30)\mathcal{U}_{[5,10]}(m = 0)$, which indicates that *both the wheels and motor control must ultimately reach* 0 *within a time frame of* 5 *to* 10 *seconds while maintaining the values no larger than* 30 *until then.*

The results of our experiment are depicted in Fig. 10. These results illustrate that during the interval from 5 to 10 seconds, the wheel (w) and motor control (m) are minimally adjusted to remain below 30, provided that these signals do not reach 0 volts.

6.2 Scalability Evaluation

To assess the scalability of our approach, we conducted experiments in which we progressively increased the complexity of the signal - specifically, the number of violation points in the signal - to examine how enforcement time is affected. The results, presented in Table 2 for the three scenarios mentioned earlier, show that the enforcement time (measured in milliseconds) increases linearly as the number of violations grows. This behavior is consistent with the predictions of our complexity analysis.

Overall, our method demonstrates robust capabilities in runtime enforcement for signals against properties specified using STL. It ensures compliance with requirements for soundness, transparency, and minimal modification across all scenarios. Moreover, it exhibits high effectiveness in managing complex signals, indicating that the time required is minimal.

7 RELATED WORKS

A framework to synthesize enforcers for reactive systems, called shields, from a set of safety properties was introduced in [6]. The uni-directional shield observes inputs from the environment and outputs from the system (program) and transforms erroneous outputs. It considered untimed properties expressed as automata.

Authors in [21, 24, 25] extendes [6] and considered bi-directional runtime enforcement for reactive systems. The enforcer presented a monitoring framework which monitors both the inputs and the outputs of a synchronous program and (minimally) edits erroneous inputs/outputs in order to guarantee a given property. In [21, 25], the properties are discrete properties, expressed using a variant of timed automata called Discrete Timed Automata (DTA) and Valued Discrete Timed Automata (VDTA). These are TAs with integer-valued clocks.

Most existing RE methods focus on properties specified by automata [22, 30]. Basin et al. proposed an RE approach for properties expressed in Metric First-Order Temporal Logic (MFOTL) [13], building upon their earlier work on runtime verification for MFOTL [4, 5]. Properties defined in MITL can also be enforced by transforming them into automata [29]. Several studies have explored the construction of automata from MITL specifications [11], and this idea has been adopted in controller synthesis methods as well [17]. Our work similarly leverages this transformation approach.

When using STL as a specification, several frameworks have been developed for monitoring STL properties. For instance, the framework in [19] automatically generates property monitors that can verify whether a given signal, with bounded length and finite variability, satisfies the specified property. However, this approach focuses on offline monitoring and does not address signal correction when the property is violated. In contrast, the authors in [33] propose enforcement (i.e., signal correction) specifically for self-driving vehicles, relying on a predictive model of the environment constructed using the vehicle's sensors.

Another approach to ensuring that a given CPS satisfies a specified STL property is controller synthesis. Raman et al. introduced a method for encoding the STL specification into MILP [27], which was subsequently used to synthesize robust controllers [28]. Additionally, Control Barrier Function-based methods have been extended to synthesize feedback controllers that respect STL specifications [16, 18]. More recently, Su et al. proposed a method for synthesizing switching controllers for CPSs that consider both continuous and discrete behaviors in relation to STL specifications [31].

Our work presents a more general approach to enforcing STL properties. Unlike existing literature, it adopts a more formal enforcement method, where the enforcer corrects the signal while adhering to some critical constraints.

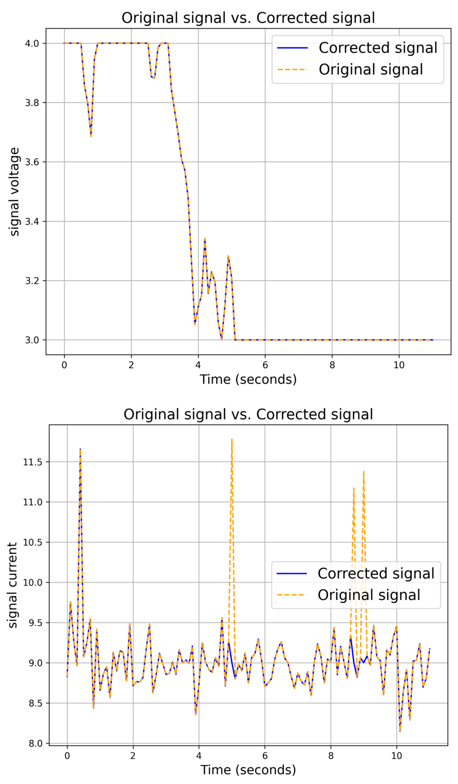


Figure 9: Enforcement of Voltage and Current Signals against Safe Charge Property

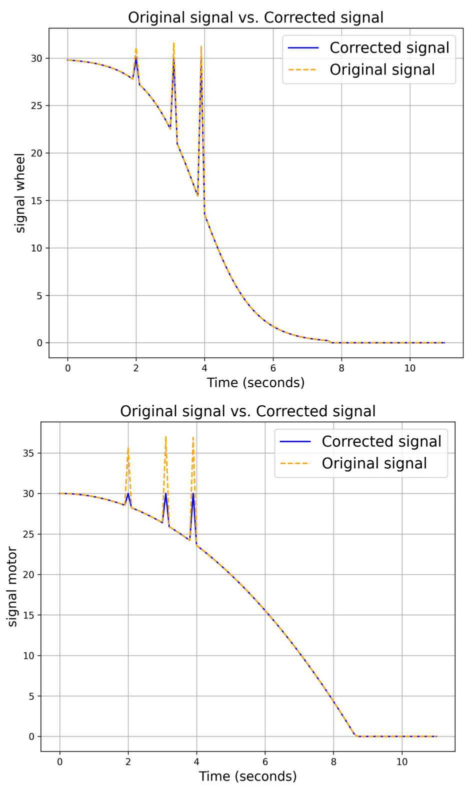


Figure 10: Enforcement of Wheel and Motor Control Signals against Safe Deceleration Property

Table 2: Experimental results with varying violation points in the signal

#v	*Safe stopping of AVs*		*Safe charging of AVs*		*Safe deceleration of AVs*	
	len(σ)	time(ms)	len(σ)	time(ms)	len(σ)	time(ms)
2	8	0.077	7	0.079	9	0.111
4	12	0.078	11	0.091	13	0.131
6	16	0.138	15	0.113	17	0.169
8	16	0.099	19	0.175	21	0.209
10	22	0.132	21	0.182	23	0.218
12	22	0.135	23	0.170	22	0.238
14	28	0.196	24	0.181	27	0.284
16	32	0.175	33	0.247	25	0.244
18	26	0.148	31	0.236	35	0.339
20	24	0.142	29	0.216	27	0.268

len(σ): the length of time word encoded from the signal; #v: the number of violation points in signal

8 CONCLUSIONS AND FUTURE WORKS

In this work, we developed a framework for the runtime enforcement against STL formula. This framework inputs a signal and outputs a minimally modified signal that satisfy the formula. Specially, given an STL formula, we derive timed transducers for the atomic components, compose them according to the formula, and apply them to the input timed words, which are obtained by encoding the signal. We present detail procedure for signal encoding, translating STL temporal operators into timed transducers, and an enforcement algorithm. Our approach effectively enforces a signal against an STL property on CPS.

As in [21, 24, 25], we plan to extend the work to accommodate bidirectionality and also extend the framework for more general STL formulas.

ACKNOWLEDGMENTS

This work has been partially funded by the NSFC under Grant No. 62192732 and 62032024, by the CAS Project for Young Scientists in Basic Research under Grant No. YSBR-040, by the CAS President's International Fellowship Initiative under Grant No. 2024PG0016.

REFERENCES

[1] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.

[2] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. 1996. The benefits of relaxing punctuality. *J. ACM* 43, 1 (Jan. 1996), 116–146. https://doi.org/10.1145/227595.227602

[3] Kyungmin Bae and Jia Lee. 2019. Bounded model checking of signal temporal logic properties using syntactic separation. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

[4] David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu. 2012. MONPOLY: Monitoring usage-control policies. In *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*. Springer, 360–364.

[5] David Basin, Felix Klaedtke, Samuel Müller, and Birgit Pfitzmann. 2008. Runtime monitoring of metric first-order temporal properties. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[6] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. 2015. Shield synthesis: Runtime enforcement for reactive systems. In *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 533–548.

[7] Georgios E Fainekos and George J Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.

[8] Yliès Falcone, Thierry Jéron, Hervé Marchand, and Srinivas Pinisetty. 2016. Runtime enforcement of regular timed properties by suppressing and delaying events. *Sci. Comput. Program.* 123, C (July 2016), 2–41. https://doi.org/10.1016/j.scico.2016.02.008

[9] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. 2011. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods Syst. Des.* 38, 3 (2011), 223–262. https://doi.org/10.1007/s10703-011-0114-4

[10] Yliès Falcone and Srinivas Pinisetty. 2019. On the runtime enforcement of timed properties. In *Runtime Verification*, Bernd Finkbeiner and Leonardo Mariani (Eds.). Springer International Publishing, Cham, 48–69.

[11] Thomas Ferrere, Oded Maler, Dejan Ničković, and Amir Pnueli. 2019. From real-time logic to timed automata. *Journal of the ACM (JACM)* 66, 3 (2019), 1–31.

[12] Ting Gan, Mingshuai Chen, Yangjia Li, Bican Xia, and Naijun Zhan. 2017. Reachability analysis for solvable dynamical systems. *IEEE Trans. Automat. Control* 63, 7 (2017), 2003–2018.

[13] François Hublet, Leonardo Lima, David Basin, Srđan Krstić, and Dmitriy Traytel. 2024. Proactive real-time first-order enforcement. In *International Conference on Computer Aided Verification*. Springer, 156–181.

[14] Jay Ligatti, Lujo Bauer, and David Walker. 2005. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.* 4, 1-2 (2005), 2–16. https://doi.org/10.1007/s10207-004-0046-8

[15] Jay Ligatti, Lujo Bauer, and David Walker. 2009. Run-time enforcement of non-safety policies. *ACM Trans. Inf. Syst. Secur.* 12, 3, Article 19 (Jan. 2009), 41 pages. https://doi.org/10.1145/1455526.1455532

[16] Lars Lindemann and Dimos V Dimarogonas. 2018. Control barrier functions for signal temporal logic tasks. *IEEE control systems letters* 3, 1 (2018), 96–101.

[17] Lars Lindemann and Dimos V Dimarogonas. 2020. Efficient automata-based planning and control under spatio-temporal logic specifications. In *2020 American Control Conference (ACC)*. IEEE, 4707–4714.

[18] Lars Lindemann, George J Pappas, and Dimos V Dimarogonas. 2020. Control barrier functions for nonholonomic systems under risk signal temporal logic specifications. In *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 1422–1428.

[19] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Yassine Lakhnech and Sergio Yovine (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 152–166.

[20] Yurii Nesterov and Arkadii Nemirovskii. 1994. *Interior-point polynomial algorithms in convex programming*. SIAM.

[21] Hammond Pearce, Srinivas Pinisetty, Partha S. Roop, Matthew M. Y. Kuo, and Abhisek Ukil. 2020. Smart I/O modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Transactions on Industrial Informatics* 16, 7 (2020), 4659–4669. https://doi.org/10.1109/TII.2019.2945520

[22] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Nguena Timo. 2014. Runtime enforcement of timed properties revisited. *Form. Methods Syst. Des.* 45, 3 (dec 2014), 381–422. https://doi.org/10.1007/s10703-014-0215-y

[23] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena Timo. 2013. Runtime enforcement of timed properties. In *Runtime Verification*, Shaz Qadeer and Serdar Tasiran (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 229–244.

[24] Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard von Hanxleden. 2017. Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 178:1–178:25. https://doi.org/10.1145/3126500

[25] Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Stavros Tripakis, and Reinhard von Hanxleden. 2017. Runtime enforcement of reactive systems using synchronous enforcers. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, Hakan Erdogmus and Klaus Havelund (Eds.). ACM, 80–89. https://doi.org/10.1145/3092282.3092291

[26] Mihai Putinar. 1993. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal* 42, 3 (1993), 969–984.

[27] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, 81–87.

[28] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*. 239–248.

[29] Matthieu Renard, Antoine Rollet, and Yliès Falcone. 2017. GREP: games for the runtime enforcement of properties. In *Testing Software and Systems: 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings 29*. Springer, 259–275.

[30] Fred B. Schneider. 2000. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.* 3, 1 (Feb. 2000), 30–50. https://doi.org/10.1145/353323.353382

[31] Han Su, Shenghua Feng, Sinong Zhan, and Naijun Zhan. 2024. Switching controller synthesis for hybrid systems against STL formulas. In *International Symposium on Formal Methods*. Springer, 229–247.

[32] Han Su, Saumya Shankar, Srinivas Pinisetty, Partha S. Roop, and Naijun Zhan. 2025. Runtime enforcement of CPS against signal temporal logic. arXiv:2502.11584 [eess.SY] https://arxiv.org/abs/2502.11584

[33] Yang Sun, Christopher M Poskitt, Xiaodong Zhang, and Jun Sun. 2024. REDriver: runtime enforcement for autonomous vehicles. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.