Automatic Trace Generation for Signal Temporal Logic

Pavithra Prabhakar
Kansas State University
Manhattan, KS, USA
pprabhakar@ksu.edu

Ratan Lal
Kansas State University
Manhattan, KS, USA
ratan@ksu.edu

James Kapinski
Toyota Technical Center
Ann Arbor MI, USA
jim.kapinski@toyota.com

***Abstract*—In this work, we present a novel technique to automatically generate satisfying and violating traces for a Signal Temporal Logic (STL) formula. STL is a logic whose formulas are interpreted over real-valued signals that evolve over dense time, which is a natural setting for Cyber-Physical Systems (CPS) applications. However, the process of developing appropriate STL requirements can be difficult and error prone. In this work, we provide a method to assist designers in the development of STL requirements for CPS applications. Our technique automatically encodes a given STL formula into a satisfiability modulo theory (SMT) formula in an appropriate theory. Satisfying and violating traces for the STL specification can be obtained by solving satisfiability problems on the encoded SMT formulas. In particular, models returned by the SMT solver correspond to traces that satisfy/violate the STL formula, thus offering a window into the types of behaviors specified by the formula. We demonstrate how the method can be used to debug problems with STL requirements, and we evaluate the performance of the method on a collection of requirements developed for CPS applications.**

***Keywords*-Signal Temporal Logic, Cyber-Physical Systems, Debugging**

I. Introduction

Cyber-physical systems (CPS) consist of software controlling physical systems, and manifest widely in automotive and aerospace industries. Many CPS applications are safety critical, and so care must be taken during the development process to make sure that expected behaviors are realized in the deployed systems. Requirements engineering is a crucial aspect of the design process that deals with defining, documenting and maintaining requirements throughout the development process. Despite this, developing requirements for CPS remains a challenging aspect of the development process, owing to certain inherent characteristics of CPS. In this work, we describe a new method to assist in the development of requirements for CPS applications. The method takes a draft requirement in the form of a temporal logic formula and automatically generates example traces that illustrate behaviors that the formula allows or prohibits. The resulting traces can be used to evaluate and debug formal requirements for CPS applications.

Pavithra Prabhakar was partially supported by NSF CAREER Award No. 1552668 and ONR YIP Award No. N00014-17-1-257. This work was partially conducted while Pavithra Prabhakar was at Toyota Technical Center.

Formal requirements are necessary to harness the power of the tools developed by the verification community, such as formal methods or requirement falsification. Despite this, formal requirements are not often written for CPS applications, for multiple reasons. One challenge is that requirements based on traditional performance metrics considered by control engineers, such as overshoots and rise times, can be difficult to capture precisely using existing requirements engineering frameworks. Another challenge is the co-design of the "physical environment", such as an engine, and the software/control algorithm. As the structure, parameters, and capabilities of the physical plant evolve during the development process, the expected behaviors of the control software (requirements) will also need to evolve. Therefore, the requirements can be expected to go through multiple revisions during the development process.

Additionally, there is the practical consideration that the formalisms used to capture requirements, such as temporal logics, are foreign to the engineers developing CPS applications, who are often not software experts. Writing specifications in formalisms such as temporal logics that capture temporal ordering of behaviors, requires expertise and training. In particular, these languages can be difficult to understand, especially, logical formulas that permit multiple levels of nesting. In addition to temporal constraints, CPS requirements have timing constraints as well as real valued signals, which make the specifications complex, as compared to specifications used for software-only systems.

To improve this situation, tools are needed to assist engineers in developing formal requirements. The approach that we use is to provide a method to take a draft requirement and automatically generate traces that satisfy or violate the requirement. The idea being that the engineer can use the examples to identify problems or otherwise unexpected aspects of the requirement. Using the insights gained from the examples, the engineer may update (improve) the draft requirement.

In one example that we present, we demonstrate how traces generated using our technique can be used to identify a significant feature of a requirement that can be easily overlooked by the requirement designer. The example considers a draft version of an *overshoot* requirement, which is intended to impose a limit on how much an output signal may surpass a given limit after a jump in the input signal. Overshoot

2576-3172/18/$31.00 ©2018 IEEE
DOI 10.1109/RTSS.2018.00038

requirements are common in CPS applications. Using our method, we demonstrate that the draft overshoot requirement can be violated by output signals that never surpass the given limit, which may be an unexpected aspect of the draft requirement. The results can be used to improve upon the draft requirement to create a requirement that better captures the designer's intention vis-à-vis the overshoot behavior.

In this paper, we focus on Signal Temporal Logic (STL) as a formal logic to capture CPS specifications, and provide a method to generate multiple satisfying and violating traces, which can aid in the process of debugging these specifications. Our broad method consists of encoding real-valued signals satisfying an STL specification using an SMT formula. More precisely, we consider piecewise linear signals with *bounded variability*, which are encoded using a finite number of variables that capture the values of the signals at a finite number of points. Our encoding allows us to obtain both satisfying and violating traces by enforcing the value of a certain boolean variable in the encoding to be true or false. A satisfying/violating trace is represented by a satisfying assignment of the SMT formula where the values of certain real valued variables capture the times of discontinuities (switching times) in the piecewise linear signals as well as the values of the signals at these switching times. The SMT solvers only return algebraic numbers as satisfying assignments, and hence, the switching times represented by non-algebraic numbers are not returned. However, this is not a serious limitation, since, if there exists a satisfying/violating trace that satisfies the formula, then there exists one with algebraic switching points. In addition, we can add constraints corresponding to a particular satisfying or violating trace to the SMT encoding as a *blocking constraint*, thereby forcing the SMT solver to return a different satisfying trace. By appropriately choosing the blocking conditions, we can eliminate traces that are either qualitatively or quantitatively different from the ones that have been generated before. Our experiments illustrate the benefits of the technique for debugging formal requirements.

Our work is relevant in the context of real-time monitoring and debugging as STL formulas can capture interesting real-time properties about signals. Since, STL essentially extends Linear Temporal Logic (LTL) and Metric Temporal Logic (MTL), it can express temporal ordering and real-time constraints. In addition, it can also express constraints on values of signals using predicates, thus capturing properties such as overshoot, rise time and other common properties of interest to control engineers. However, it cannot express properties such as stability, that refer to multiple traces and/or require metrics to express convergence. An interesting future direction will be to explore extensions of the logic such as Hyper Signal Temporal Logics along the linear of HyperLTL [6], [19] to capture properties about multiple traces. It can express temporal ordering and real-time constraints (for details on expressiveness, see [12], [20]).

II. Related work

STL [11] has gained attention in the recent years as a formal logic for specifying properties of CPS. Model-checking [23] techniques for STL have been investigated, and efficient monitoring algorithms have been developed [8], [13]. Also, control synthesis methods based on STL specification have been proposed [15], [21]. Trajectory generation based on optimization has been addressed in [24] for linear temporal logic (LTL). Satisfiability checking has been investigated extensively for LTL [2], [18]. Metric temporal logic (MTL) and its satisfiability have been investigated in [3]–[5], [17]; however, the work on generating *multiple* satisfying/violating traces for a specification is limited, especially for STL, which is closely related to MTL. The closest work related to ours is the tool STLInspector [22], which generates multiple traces from STL specifications. To the best of our knowledge, the theory behind STLInspector is not published and hence, unavailable for comparison; however, the tool is available for experimentation, and our preliminary studies indicate that we can consider a richer class of predicates, namely, linear, whereas STLInspector only allows rectangular predicates (variables compared with constants). Another highlight of our approach in comparison to STLInspector and alternate methods that use encoding of STL into some constraint solving formalism [21], [24] is that we output signals that are represented by their values at varying time intervals, as opposed to representations over fixed time intervals. This could result in significantly lower number of variables in the constraint solving problem, though the encoding is more complex to understand.

III. Preliminaries

Let $\mathbb{R}$ denote the set of real numbers, $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers, $\mathbb{Z}$ the set of integers and $\mathbb{B}$ the set of boolean values, namely, $\{\top, \bot\}$.

Definition 1: A *signal* over a set A is a function $\sigma : \mathbb{R}_{\geq 0} \to A$. If A is $\mathbb{B}$, then we refer to σ as a boolean signal.

We will interpret a signal as assigning a value from a set A to every time point in $\mathbb{R}_{\geq 0}$. In the sequel, we will consider certain classes of signals that can be represented by specifying their values at a finite number of time points. In particular, we consider two interpretations of the finite sequence of values, namely, as piecewise constant signals and piecewise linear signals.

Definition 2: A *boolean timed trace* of length k is a sequence $\tau = (a_1, t_1, b_1)\ (a_2, t_2, b_2) \cdots (a_k, t_k, b_k)$ in $(\mathbb{B} \times \mathbb{R}_{\geq 0} \times \mathbb{B})^k$, such that $t_1 = 0$, and $t_i < t_{i+1}$ for all $i < k$.

Definition 3: Given a boolean timed trace $\tau = (a_1, t_1, b_1)(a_2, t_2, b_2) \cdots (a_k, b_k, t_k)$, we associate a piecewise constant signal $pwc(\tau) : \mathbb{R}_{\geq 0} \to \mathbb{B}$ with it, defined as, $pwc(\tau)(t) = a_i$ if $t = t_i$ for each i, and $pwc(\tau)(t) = b_i$ if $t \in (t_i, t_{i+1})$ for each i.

Figure 1 shows a boolean timed trace τ of length 5. Consider the case where $a_1 = b_1 = a_2 = b_2 = \bot$ and $a_3 = b_3 = a_4 = b_4 = a_5 = \top$. The boolean signal $pwc(\tau)$ has value $\bot$ in the interval $[t_1, t_3)$ and value $\top$ in the interval $[t_3, \infty)$.

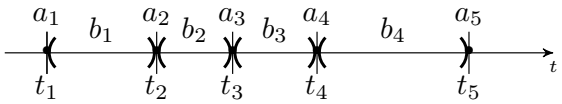

Fig. 1. Timed trace

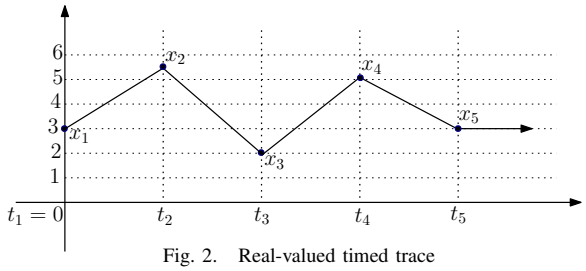

Fig. 2. Real-valued timed trace

Definition 4: A *real-valued timed trace* of length k is a sequence $\tau = (x_1, t_1)(x_2, t_2)\cdots(x_k, t_k)$ in $(\mathbb{R}^n \times \mathbb{R}_{\geq 0})^k$, such that $t_1 = 0$, and $t_i < t_{i+1}$ for all $i < k$.

Definition 5: A real-valued signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is called a *piecewise linear signal* if there exist $t_1 = 0 < t_2 < \ldots < t_k$ such that $\sigma(t) = \sigma(t_i) + \frac{\sigma(t_{i+1}) - \sigma(t_i)}{t_{i+1} - t_i}(t - t_i)$ if $t \in [t_i, t_{i+1})$ for some $1 \leq i < k$, and $\sigma(t) = \sigma(t_k)$, if $t \geq t_k$.

Definition 6: Given a real-valued timed trace $\tau = (x_1, t_1)(x_2, t_2)\cdots(x_k, t_k)$, we associate a piecewise linear signal $pwl(\tau) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ with it, defined as, $pwl(\tau)(t) = x_i + \frac{(x_{i+1} - x_i)}{t_{i+1} - t_i}(t - t_i)$ if $t \in [t_i, t_{i+1})$ for some $1 \leq i < k$, and $pwl(\tau)(t) = x_k$ for all $t \geq t_k$.

Figure 2 shows a real-valued timed trace τ of length 5. Consider the case where $x_1 = 3,\ x_2 = 5.5,\ x_3 = 2,\ x_4 = 5,\ x_5 = 3,\ t_1 = 0,\ t_2 = 1,\ t_3 = 2,\ t_4 = 3$ and $t_5 = 4$. The piecewise linear signal corresponding to the real-valued timed trace is $pwl(\tau)(t_1) = 3,\ pwl(\tau)(t_2) = 5.5,\ pwl(\tau)(t_3) = 2,\ pwl(\tau)(t_4) = 5,\ pwl(\tau)(t_5) = 3$ and

$$pwl(\tau)(t) = \begin{cases} 3 + 2.5t, \text{ if } t \in (0,1) \\ 5.5 - 3.5(t-1), \text{ if } t \in (1,2) \\ 2 + 3(t-2), \text{ if } t \in (2,3) \\ 5 - 2(t-3), \text{ if } t \in (3,4) \\ 3, \text{ if } t > 4. \end{cases}$$

A. Linear Predicate

A linear predicate over real variables $\mathbf{x} = (x_1, \ldots, x_n)$ denoted by $p(\mathbf{x})$, is a linear constraint, that is, $c_1x_1 + \ldots + c_nx_n \sim d$, where $c_i \in \mathbb{Z}$, for $1 \leq i \leq n$, $d \in \mathbb{Z}$ are constants, and $\sim \in \{<, \leq\}$. We write $c_1x_1 + \ldots + c_nx_n \sim d$ as $c\mathbf{x} \sim d$, where $c = (c_1, \ldots, c_n)$ is a constant vector. A valuation for $\mathbf{x}$ is a real vector $\mathbf{v} = (v_1, \ldots, v_n)$, where we interpret $v_i \in \mathbb{R}$ as an assignment for x_i. Let $p(\mathbf{v})$ denote the constraint obtained by replacing x_i with v_i, for every i. Let $[\![p]\!]$ denote the set of all valuations for $\mathbf{x}$ that satisfy the linear predicate p, that is, $[\![p]\!] = \{\mathbf{v} \mid p(\mathbf{v}) \text{ is true}\}$. A closure of a linear predicate $p(\mathbf{x})$ denoted by $Cl(p)(\mathbf{x})$, is a linear predicate such that $[\![Cl(p)]\!]$ includes all limit points of sequences in the set represented by linear predicate p, that is, $Cl(p)(\mathbf{x})$ is $c\mathbf{x} \leq d$, where $p(\mathbf{x})$ is $c\mathbf{x} \sim d$. The next two lemmas state properties about linear predicates.

Lemma 1: Given a linear predicate $p(\mathbf{x})$ over the variables $\mathbf{x} = (x_1, \ldots, x_n)$, and two points $\mathbf{v}$, $\mathbf{v}' \in \mathbb{R}^n$,

1) if $\mathbf{v}, \mathbf{v}' \in [\![p]\!]$ then $\alpha\mathbf{v} + (1-\alpha)\mathbf{v}' \in [\![p]\!]$, $\forall\ 0 \leq \alpha \leq 1$.
2) if $\mathbf{v} \in [\![p]\!]$ and $\mathbf{v}' \in [\![Cl(p)]\!]$ then $\alpha\mathbf{v} + (1-\alpha)\mathbf{v}' \in [\![p]\!]$, for all $0 < \alpha \leq 1$.

The first part of Lemma 1 states that if two points are contained in a half-space, then all point on the line segment joining the two points will also lie in the half space. The second part states that if one of the points is contained in a half-space, while the other is on the boundary (the latter point might not be in the half-space if the corresponding constraint is a strict inequality), then still all points on the line segment joining the two points will be in the half-space except possibly for the end-point which is on the boundary.

IV. Signal Temporal Logic

In this section, we discuss signal temporal logic (*STL*) [9] which is a formal language for specifying a set of signals. *STL* extends linear temporal logic (LTL) in two directions: it allows timing intervals to be associated with temporal operators (as in metric temporal logic), and atomic propositions to be specified using predicates over real valued variables. The formulae φ of *STL* over variables $\mathbf{x} = (x_1, \cdots, x_n)$ are defined inductively as: $\varphi ::= p(\mathbf{x}) \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi U_I \varphi$,

where I is a closed interval over the non-negative reals, and p is a linear predicate over $\mathbf{x}$.

An *STL* formula φ is interpreted over a signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$. Given a signal σ and time $t \in \mathbb{R}_{\geq 0}$, we define the satisfaction of a formula φ at time t in σ, denoted $\sigma, t \models \varphi$, inductively as follows:

- $\sigma, t \models p$ if and only if $p(\sigma(t)) = \top$.
- $\sigma, t \models \neg\varphi$ if and only if it is not the case that $\sigma, t \models \varphi$, that is, $\sigma \not\models \varphi$.
- $\sigma, t \models \varphi_1 \vee \varphi_2$ if and only if $\sigma, t \models \varphi_1$ or $\sigma, t \models \varphi_2$.
- $\sigma, t \models \varphi_1 U_I \varphi_2$ if and only if there exists $t' \geq t$ such that $t' - t \in I$, $\sigma, t' \models \varphi_2$ and for all $t'' \in [t, t']$, $\sigma, t'' \models \varphi_1$.

We say that σ *satisfies* φ, denoted $\sigma \models \varphi$, if $\sigma, 0 \models \varphi$; and σ *violates* φ, if $\sigma \not\models \varphi$.

Let $\Diamond_I \varphi \equiv \top U_I \varphi$ and $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$. Note that then $\Diamond_I \varphi \equiv \neg\Box_I \neg\varphi$. We can write U_I using exclusively $U_{[0,\infty)}$ (also represented as just U) and $\Box_I \varphi$ temporal operators as follows which is similar to until rewrite in the paper [10].

$$\varphi_1 U_I \varphi_2 = \Box_{[0,a]}\varphi_1 \wedge \Diamond_I \varphi_2 \wedge \Diamond_{[a,a]}\varphi_1 U \varphi_2,$$

where $I = [a, b]$ or $[a, \infty)$. From now on, we will assume that an *STL* formula is given to us using only U and $\Box_I$ as the temporal operators, and $\vee$ and $\neg$ as the boolean operators.

A. Finite Variability

Our objective is to find signals that satisfy and violate a given *STL* formula. We focus on a subclass of signals with restricted variability. Let $SF(\varphi)$ represent the set of all sub-formulas of a formula φ. It is defined inductively as follows:

- $SF(p) = \{p\}$.
- $SF(\neg\varphi) = SF(\varphi) \cup \{\neg\varphi\}$.
- $SF(\varphi_1 \vee \varphi_2) = SF(\varphi_1) \cup SF(\varphi_2) \cup \{\varphi_1 \vee \varphi_2\}$.
- $SF(\Box_I \varphi) = SF(\varphi) \cup \{\Box_I \varphi\}$.
- $SF(\varphi_1 U \varphi_2) = SF(\varphi_1) \cup SF(\varphi_2) \cup \{\varphi_1 U \varphi_2\}$.

Given a signal σ and a formula φ, we are interested in understanding how the satisfiability of φ varies with time in σ. We use σ^φ to represent a boolean signal that captures whether or not φ is satisfied at different time points.

Definition 7: Given a signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and an *STL* formula φ, we define a boolean signal $\sigma^\varphi : \mathbb{R}_{\geq 0} \to \mathbb{B}$, where $\sigma^\varphi(t) = \top$ if and only if $\sigma, t \models \varphi$.

Definition 8: A boolean signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{B}$ is *finitely varying* if there exist times $0 = t_1 < t_2 < \ldots < t_k < t_{k+1} = \infty$, such that for each $0 \leq i \leq k$, there exists boolean constant c such that $\sigma(t) = c$, for $t \in (t_i, t_{i+1})$.

The following lemma states that the satisfaction of an *STL* formula over time in a signal σ is finitely varying if σ is a piecewise linear signal.

Lemma 2: If σ is a piecewise linear signal and φ is an *STL* formula, then σ^φ is finitely varying.

We intend to generate piecewise linear signals that change the satisfaction of any sub formula of a give formula at most k times.

Definition 9: Given a piecewise linear signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and an *STL* formula φ over $\mathbf{x}$, we say that σ is k-varying with respect to φ, if there exist times $0 = t_1 < t_2 < \ldots < t_k < t_{k+1} = \infty$, such that for each sub-formula $\psi \in SF(\varphi)$, the discontinuous points of signal σ^ψ and the points of non-differentiability of σ are contained in $\{t_1, \ldots, t_k\}$. We refer to each of the t_is as a (possible) switching time.

Consider an *STL*-formula φ which is given by the predicate $x \geq 3$. The satisfaction of φ in the timed trace τ in Figure 2 changes at times $12/7$ and $7/3$, though the points of non-differentiability of σ are $0, 1, 2, 3$ and 4. Hence, τ is 7-varying with respect to φ. The times of non-differentiability of a signal can be very different from the time points at which the satisfiability of a formula changes, as in the above example. However, the following lemma states that if σ is a piecewise linear signal, then there is a bound on the number of discontinuities of σ^ψ for any sub formula ψ of σ.

Lemma 3: If σ is a piecewise linear signal and φ is an *STL* formula, then there exists k such that σ is k-varying with respect to φ.

B. Satisfiability Modulo Theories and Trace Generation Problem

We present a brief introduction to satisfiability modulo theory (SMT) [1] restricted to our context. The class of SMT formulae we consider are a subclass of STL formulae where the timed until operator is not allowed. The SMT formula φ is defined over variables $\mathbf{x} = (x_1, \ldots, x_n)$ inductively as: $\varphi ::= p(\mathbf{x}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$, where $p(\mathbf{x})$ is a linear predicate. Here we focus on SMT formulae in the theory of linear real arithmetic, that is, the formulae are interpreted over reals. Given an SMT formula, the problem of checking whether the formula is satisfiable (does there exist real values for the variables such that the formula evaluates to true) is decidable, and there are tools such as Z3 [7] which can perform satisfiability checking and return a satisfying assignment.

Our main goal is to develop a technique to automatically generate satisfying and violating traces for an *STL* formula, to aid the user with debugging formal requirements. The problem is, in general, undecidable, given the undecidability results for the sub-logic MTL (metric temporal logic) [14]. Hence, we focus on finding traces with finite variability, which is similar in spirit to bounded verification. We restrict ourselves to piecewise linear signals, however, our broad framework is applicable to generate other classes of signals as well. We will need to develop the specifics of extracting a signal of a certain kind (for instance, piecewise constant or differentiable) from a boolean signal that captures the satisfaction of predicates.

The main focus of this work will be to address the problem of generating a single satisfying or violating trace. We will briefly discuss some heuristics that will generate a sequence of satisfying/violating traces; however, detailed investigation of the same is left for future work.

Problem 1 (Satisfying trace): Given an *STL* formula φ over $\mathbf{x} = (x_1, \ldots, x_n)$, find a k-varying piecewise linear signal $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, such that $\sigma \models \varphi$.

V. Encoding

In this section, we present an approach to solve Problem 1 by reducing it to satisfiability solving of an SMT formula. Our broad approach consists of constructing an SMT formula $Enc(\varphi, k)$ over a set of variables that capture the values of σ^ψ for every $\psi \in SF(\varphi)$ at the switching points of σ with respect to φ. The set of variables of the formula $Enc(\varphi, k)$, denoted $Var(\varphi, k)$, is the union of the following sets:

- $\{t_1, \ldots, t_k\}$, where t_i denotes the i-th switching time instant. We will use the notation t_{k+1} to denote ∞.
- $\{a_i^{\mathbf{x}} \mid 1 \leq i \leq k\}$, where $a_i^{\mathbf{x}} = (a_i^{x_1}, \ldots, a_i^{x_n})$ is an n-tuple of real-valued variables that captures the value of $\mathbf{x}$ at the i-th switching time.
- $\{a_i^\psi, b_i^\psi \mid \psi \in SF(\varphi), 1 \leq i \leq k\}$, where a_i^ψ and b_i^ψ are two boolean variables that capture the (boolean) value of σ^ψ at time t_i and in the interval (t_i, t_{i+1}).
- $\{P_i^\psi, C_i^\psi \mid \Box_I \psi \in SF(\varphi), 1 \leq i \leq k\}$, where P_i^ψ is an auxiliary real valued variable that captures the longest time period immediately prior to the i^{th} switching time instant t_i for which the formula φ is true, and C_i^ψ a boolean variable that captures whether φ is true at $t_i - P_i^\psi$.

Next, we provide the construction of $Enc(\varphi, k)$ and theorems for its correctness.

A. Construction of $Enc(\varphi, k)$

The construction of $Enc(\varphi, k)$ is defined inductively using an additional formula $Form(\varphi, k)$, which captures some constraints corresponding to the highest level connective in φ, as follows:

- $Enc(p, k) = Form(p, k)$.
- $Enc(\neg\varphi, k) = Enc(\varphi, k) \wedge Form(\neg\varphi, k)$.

- $\mathit{Enc}(\varphi_1 \vee \varphi_2, k) = \mathit{Enc}(\varphi_1, k) \ \wedge \mathit{Enc}(\varphi_2, k) \wedge \mathit{Form}(\varphi_1 \vee \varphi_2, k)$.
- $\mathit{Enc}(\Box_I \varphi, k) = \mathit{Enc}(\varphi, k) \wedge \mathit{Form}(\Box_I \varphi, k)$.
- $\mathit{Enc}(\varphi_1 U \varphi_2, k) = \mathit{Enc}(\varphi_1, k) \wedge \mathit{Enc}(\varphi_2, k) \ \wedge \mathit{Form}(\varphi_1 U \varphi_2, k)$.

We will not explicitly provide the proof for the inductive step of Lemma 4, but explain the intuition behind our construction. Next, we provide the construction of $\mathit{Form}(\varphi, k)$ by induction on the structure of φ.

1) Case $\varphi = p(\mathbf{x})$: We will add constraints that will ensure that the piecewise linear signal corresponding to $\tau^{\mathbf{v},\mathbf{x}}$ and the boolean piecewise constant signal corresponding to $\tau^{\mathbf{v},p}$ are consistent. First, we add the timing requirements:

$$t_1 = 0 \wedge \bigwedge_{1 \leq i < k} t_i < t_{i+1} \tag{1}$$

Next, we observe using Lemma 1 that to check that a linear signal satisfies a predicate (everywhere) in the interval (t_i, t_{i+1}), that is, $b_i^p = \top$, it suffices to check that both its end-points satisfy the closure of the predicate, and that at least one of the end-points satisfies the predicate. Similarly, for $b_i^p = \bot$, we need to ensure that the linear signal violates the predicate (everywhere) in the interval (t_i, t_{i+1}). Note that we are only encoding k-varying signals, which means that the satisfaction of p only changes at the time instants t_i. Hence, we have the following relation between the boolean variables for p and the real-valued variables for $\mathbf{x}$.

$$\{b_k^p = a_k^p \wedge \bigwedge_{1 \leq i \leq k} \{a_i^p = p(a_i^{\mathbf{x}}) \wedge \tag{2}$$

$$\{b_i^p \Rightarrow \{[p(a_i^{\mathbf{x}}) \wedge Cl(p)(a_{i+1}^{\mathbf{x}})] \vee [Cl(p)(a_i^{\mathbf{x}}) \wedge p(a_{i+1}^{\mathbf{x}})]\}\} \wedge \tag{3}$$

$$\begin{aligned} \{\neg b_i^p \Rightarrow \{[\neg p(a_i^{\mathbf{x}}) \wedge Cl(\neg p)(a_{i+1}^{\mathbf{x}})] \vee \\ [Cl(\neg p)(a_i^{\mathbf{x}}) \wedge \neg p(a_{i+1}^{\mathbf{x}})]\}\}\} \end{aligned} \tag{4}$$

$\mathit{Form}(p, k)$ is the conjunction of the formulas (1), (2), (3), and (4).

2) Case $\varphi = \neg\varphi_1$: $\mathit{Form}(\varphi, k)$ relates the boolean variables for φ and φ_1, and is given by:

$$\bigwedge_{1 \leq i \leq k} [(a_i^{\varphi} = \neg a_i^{\varphi_1}) \wedge (b_i^{\varphi} = \neg b_i^{\varphi_1})] \tag{5}$$

3) Case $\varphi = \varphi_1 \vee \varphi_2$: $\mathit{Form}(\varphi, k)$ relates the boolean variables for φ_1 and φ_2 with that for φ, and is given by:

$$\bigwedge_{1 \leq i \leq k} (a_i^{\varphi} = (a_i^{\varphi_1} \vee a_i^{\varphi_2})) \wedge (b_i^{\varphi} = (b_i^{\varphi_1} \vee b_i^{\varphi_2})) \tag{6}$$

4) Case $\varphi = \varphi_1 U \varphi_2$: $\mathit{Form}(\varphi, k)$ relates the boolean variables for φ_1 and φ_2 with that for φ, and is given by the conjunction of formulas 7 and 8.

$$\bigwedge_{1 \leq i \leq k} [a_i^{\varphi} = ((a_i^{\varphi_1} \wedge a_i^{\varphi_2}) \vee (a_i^{\varphi_1} \wedge b_i^{\varphi}))] \tag{7}$$

If $\varphi_1 U \varphi_2$ is true at t_i, then either φ_1 and φ_2 both hold at t_i, or φ_2 holds at some point in the future and φ_1 holds until then. Note that even if φ_2 holds immediately after t_i, say, in the interval (t_i, t_{i+1}), but φ_1 does not hold in (t_i, t_{i+1}), then $\varphi_1 U \varphi_2$ will not hold at t_i, even if φ_1 holds at t_i. This is because (t_i, t_{i+1}) is open on the left, and hence, choosing any t in that interval to satisfy φ_2, will require φ_1 to be true in the interval $[t_i, t]$, which is non-empty, and hence, φ_1 needs to be true in the interval (t_i, t_{i+1}) because of the finite variability of the signal with respect to any sub formula of φ.

$$\bigwedge_{1 \leq i \leq k} [b_i^{\varphi} = ((b_i^{\varphi_1} \wedge b_i^{\varphi_2}) \vee (b_i^{\varphi_1} \wedge a_{i+1}^{\varphi}))]\} \wedge b_k^{\varphi} = b_k^{\varphi_1} \wedge b_k^{\varphi_2}] \tag{8}$$

Note that the satisfaction of $\varphi_1 U \varphi_2$ in the interval (t_i, t_{i+1}) is possible only if both φ_1 and φ_2 hold in that interval or φ_1 holds in that interval and φ holds at t_{i+1}.

5) Case $\varphi = \Box_I \varphi_1$: The formula $\mathit{Form}(\varphi, k)$ needs to ensure that the valuations for the boolean variables corresponding to φ and φ_1 are consistent. First, note that for φ to be true to t_i, φ_1 needs to be true everywhere in the interval $t_i + I$, which is equivalent to checking that φ_1 needs to hold at all switching times t_j that occur within $t_i + I$, and in all intervals (t_j, t_{j+1}) that overlap with $t_i + I$. This property is captured using the formula $\mathbb{C}(J)$, parameterized by an interval J as follows.

$$\bigwedge_{1 \leq j \leq k} ((t_j \in J \Rightarrow a_j^{\varphi_1} = 1) \wedge ((t_j, t_{j+1}) \cap J \neq \emptyset \Rightarrow b_j^{\varphi_1} = 1))$$

Similarly, if φ doesn't hold at t_i, then we require that φ_1 does not hold everywhere in the interval $t_i + I$, or it satisfies $\neg\mathbb{C}(t_i + I)$. Hence, we have:

$$\bigwedge_{1 \leq i \leq k} (a_i^{\varphi} \Leftrightarrow \mathbb{C}(t_i + I)) \tag{9}$$

Again, for φ to hold in the interval (t_i, t_{i+1}) we require that φ_1 hold everywhere in the interval $(t_i, t_{i+1}) + I$, that is, $\mathbb{C}((t_i, t_{i+1}) + I)$ holds. However, for φ to be false everywhere in the interval, that is, to assign b_i^{φ} to be false, it does not suffice if $\neg\mathbb{C}((t_i, t_{i+1}) + I)$ holds. In other words, if φ_1 doesn't hold at some point in the interval $(t_i, t_{i+1}) + I$, it does not necessarily violate $\Box_I \varphi_1$ at all points in (t_i, t_{i+1}). Instead, we require that for every $t \in (t_i, t_{i+1})$, the interval $t + I$ violates φ_1 at some point. The formula $\mathbb{N}(J_0, J_1)$ parameterized by intervals J_0 and J_1 captures the property that for every $t \in J_0$, φ_1 is violated at least once within the interval $t + J_1$. We will formally define $\mathbb{N}(J_0, J_1)$ next, however, first we complete the construction for $\mathit{Form}(\varphi, k)$ assuming that we have a construction for $\mathbb{N}(J_0, J_1)$.

$$\bigwedge_{1 \leq i \leq k} ((b_i^{\varphi} \Rightarrow \mathbb{C}((t_i, t_{i+1}) + I)) \wedge (\neg b_i^{\varphi} \Rightarrow \mathbb{N}((t_i, t_{i+1}), I)) \tag{10}$$

To define $\mathbb{N}$, let us first consider the case where $J_0 = [\mathbf{l}_0, \infty)$ and $J_1 = [\mathbf{l}_1, \infty)$ are unbounded. We need to encode that φ_1 is violated in every interval $t + J_1$, where $t \in J_0$. This forces φ_1 to be false at arbitrarily large times, hence, $\mathbb{N}(J_0, J_1)$ in this case is true only if $b_k^{\varphi_1} = \top$. This is the case for $\mathbb{N}((t_k, t_{k+1}), I)$, where t_{k+1} is defined to be ∞.

Next, let us consider the case where $J_0 = (\mathbf{l}, \mathbf{u})$ is bounded and $J_1 = [\mathbf{l}_1, \infty)$ is unbounded. If φ_1 is false somewhere in the interval $[\mathbf{u} + \mathbf{l}_1, \infty)$, then it serves as a violating witness for every interval $t + J_1$, where $t \in J_0$. If that is not the case, then the violating witness for all points in $[\mathbf{u}, \mathbf{l}]$ lie in the interval

$(\mathbf{l}+\mathbf{l}_1, \mathbf{u}+\mathbf{l}_1)$. In particular, there have to be violations of φ_1 arbitrarily close to $\mathbf{u}+\mathbf{l}_1$, otherwise some point t close to $\mathbf{u}$ will not have a violating instance of φ_1 in $(t+\mathbf{l}_1, \mathbf{u}+\mathbf{l}_1)$, note, however, due to the unboundedness of J_1, if φ_1 is violated in some open interval $(\mathbf{u}+\mathbf{l}_1-\epsilon, \mathbf{u}+\mathbf{l})$ for $\epsilon > 0$, then it serves a a witness for all the points in J_0. The following formula encodes the above observation, and defines $\mathbb{N}(J_0, J_1)$ for this case.

$$\bigvee_{1\leq j\leq k} (t_j \in [\mathbf{u}+\mathbf{l}_1, \infty) \wedge a_j^{\varphi_1} = \bot) \vee \bigvee_{1\leq j\leq k} (t_{j+1} \in [\mathbf{u}+\mathbf{l}_1, \infty) \wedge b_j^{\varphi_1} = \bot) \tag{11}$$

Finally, let us consider the case where $J_1 = [\mathbf{l}, \mathbf{u}]$ is bounded. Then $\mathbb{N}(J_0, J_1)$ is equivalent to checking if every $t + J_1$ contained in the interval $J_0 + J_1$ violates φ_1 at some point. Hence, we define a predicate $\mathbb{D}(J_1, J_2)$ which captures the fact that φ_1 is violated in every interval $t + J_1$, which is contained in J_2. It is easy to observe that $\mathbb{N}(J_0, J_1)$ is equivalent to $\mathbb{D}(J_1, J_0 + J_1)$.

To define $\mathbb{D}(J_1, J_2)$ we use two auxiliary variables, a real-valued variable $P_i^{\varphi_1}$ that captures the longest time period immediately prior to the i-th switching time t_i for which the formula φ_1 was true, and a boolean variable $C_i^{\varphi_1}$ to denote if φ_i was true at $t_i - P_i^{\varphi_1}$. For example, if the longest period prior to t_i in which φ_1 was true is $[t_j, t_i)$, then $P_i^{\varphi_1} = t_i - t_j$, and $C_i^{\varphi_1} = \top$ since $[t_j, t_i)$ is left closed. If instead the longest period prior to t_i in which φ_1 was true is (t_j, t_i), then $P_i^{\varphi_1} = t_i - t_j$ as before, but $C_i^{\varphi_1} = \bot$ since (t_j, t_i) is left open. Note $P_i^{\varphi_1}$ and $C_i^{\varphi_1}$ do not depend on the value of φ_1 at t_i itself, but on the time before that. Note that at $t_1 = 0$, there are no time points before t_1. Hence, we have

$$(P_1^{\varphi_1} = 0 \wedge C_1^{\varphi_1} = \bot) \tag{12}$$

Assume that we have appropriately set the values of $P_{i-1}^{\varphi_1}$ and $C_{i-1}^{\varphi_1}$. Note that the values of $P_i^{\varphi_1}$ and $C_i^{\varphi_1}$ depend on the value of φ_1 in the interval $[t_{i-1}, t_i)$. In particular, if φ_1 is false (everywhere) in (t_{i-1}, t_i), then $P_i^{\varphi_1} = 0$ and $C_i^{\varphi_1} = \bot$ irrespective of the value of φ_1 at t_{i-1}. If φ_1 is true in (t_{i-1}, t_i), but false at t_{i-1}, then $P_i^{\varphi_1} = t_i - t_{i-1}$, the longest period for which φ_1 was true continuously prior to t_i, and $C_i^{\varphi_1} = \bot$ because the longest interval (t_{i-1}, t_i) is open. Finally, if φ_1 is true in the interval $[t_{i-1}, t_i)$, then $P_i^{\varphi_1} = t_i - t_{i-1} + P_{i-1}^{\varphi_1}$, since the longest period for which φ_1 is true will continue backward from t_i (past t_{i-1}) until the beginning of the starting point of the longest interval corresponding to $P_{i-1}^{\varphi_1}$. Hence, $C_i^{\varphi_1}$ will be the same as that for $C_{i-1}^{\varphi_1}$. These constraints are summarized below:

$$(P_1^{\varphi_1} = 0 \wedge C_1^{\varphi_1} = \bot) \tag{13}$$

$$\bigwedge_{1<i\leq k} \{b_{i-1}^{\varphi_1} = \bot \Rightarrow (P_i^{\varphi_1} = 0 \wedge C_i^{\varphi_1} = \bot)\} \tag{14}$$

$$\bigwedge_{1<i\leq k} \{(b_{i-1}^{\varphi_1} = \top \wedge a_{i-1}^{\varphi_1} = \bot) \Rightarrow (P_i^{\varphi_1} = t_i - t_{i-1} \wedge C_i^{\varphi_1} = \bot)\} \tag{15}$$

$$\bigwedge_{1<i\leq k} \{(b_{i-1}^{\varphi_1} = a_{i-1}^{\varphi_1} = \top) \Rightarrow [(P_i^{\varphi_1} = t_i - t_{i-1} + P_{i-1}^{\varphi_1}) \wedge (C_i^{\varphi_1} = C_{i-1}^{\varphi_1})]\} \tag{16}$$

Now that we have added constraints to instantiate the auxiliary variables $P_i^{\varphi_1}$ and $C_i^{\varphi_1}$, appropriately, we proceed to explain the other constraints that need to be added to $\mathbb{D}(J_1, J_2)$. Intuitively, we will check using the variable $P_i^{\varphi_1}$, if there is a period time of the length of the interval J_1 within J_2 for which φ_1 was true.

Let $J_1 = [\mathbf{l}, \mathbf{u}]$, for $\mathbf{l}, \mathbf{u} \in \mathbb{Z}$, and let $J_2 = (\mathbf{L}, \mathbf{U})$, where $\mathbf{L} \in \mathbb{Z}$ and $\mathbf{U} \in \mathbb{Z} \cup \{\infty\}$, and $\mathbf{U} - \mathbf{L} > \mathbf{u} - \mathbf{l}$, and the size of J_2 is bigger than that of J_1. If the last switching point t_k is before the beginning of J_2, then φ_1 has to be false in the interval $[t_k, \infty)$, otherwise, φ_1 will be true everywhere in $J_2 \subseteq [t_k, \infty)$, and J_2 will contain an interval of size J_1 in which φ_1 is true. Hence,

$$t_k \leq \mathbf{L} \Rightarrow a_k^{\varphi_1} = \bot \tag{17}$$

Next, we check that for every switching point t_j that lies in J_2, φ_1 is not always true in $[t_j - (\mathbf{u}-\mathbf{l}), t_j]$, the interval preceding t_j whose size is that of J_1, provided that $[t_j - (\mathbf{u}-\mathbf{l}), t_j]$ is in J_2. Hence, we check that if t_j is in J_2 and is at least at distance $\mathbf{u}-\mathbf{l}$ from the lower bound of J_2, namely, $\mathbf{L}$, then $P_j^{\varphi_1} < \mathbf{u}-\mathbf{l}$, the longest interval immediately before t_j where φ_1 is satisfied has length strictly less than $\mathbf{u}-\mathbf{l}$, or $P_j^{\varphi_1} = \mathbf{u}-\mathbf{l}$, but φ_1 is not satisfied at one of the end points of that longest interval, that is, $C_j^{\varphi_1} = \bot \vee a_j^{\varphi_1} = \bot$.

$$\bigwedge_{1\leq j\leq k} \{((t_j \in J_2) \wedge (t_j - \mathbf{L} > \mathbf{u}-\mathbf{l})) \Rightarrow (P_j^{\varphi_1} < \mathbf{u}-\mathbf{l} \vee (P_j^{\varphi_1} = \mathbf{u}-\mathbf{l} \wedge (C_j^{\varphi_1} = \bot \vee a_j^{\varphi_1} = \bot)))\} \tag{18}$$

Let t_j be the last switching point before $\mathbf{U}$. It could happen that φ_1 is satisfied (everywhere) in the interval $[t_j, t_{j+1})$, thus, creating an interval of size $\mathbf{u}-\mathbf{l}$ within J_2 where φ_1 is always satisfied. Let us consider a couple of cases:

- Case $j < k$ (encoded in (19)): We need to ensure that $P_{j+1}^{\varphi_1}$ minus the length of the interval $[\mathbf{U}, t_{j+1}]$, which represents the longest interval prior to $\mathbf{U}$ where φ_1 is true, is at most $\mathbf{u}-\mathbf{l}$.
- Case $j = k$ and $a_j^{\varphi_1}$ and $b_j^{\varphi_1}$ are true (encoded in (20)): Note that in this case if $P_j^{\varphi_1} \geq t_j - \mathbf{L}$, then φ_1 is true in the interval $(\mathbf{L}, t_j)$, and φ_1 is true in the interval $[t_j, \mathbf{U})$ (because of the current case we are considering), we obtain that φ_1 is true in the whole interval $(\mathbf{L}, \mathbf{U})$ whose size is larger than $(\mathbf{l}, \mathbf{u})$, hence, $\mathbb{D}(J_1, J_2)$ will be false. Hence, we require $P_j^{\varphi_1} < t_j - \mathbf{L}$ and sum of $P_j^{\varphi_1}$ and $\mathbf{U} - t_j$ to be less than or equal to $\mathbf{u}-\mathbf{l}$.
- Case $j = k$ and $a_k^{\varphi_1}$ is false and $b_k^{\varphi_1}$ is true (encoded in (21)): Here, we require that the length of the interval $[t_j, \mathbf{U})$ is less than $\mathbf{u}-\mathbf{l}$.
- Case $j = k$ and $a_k^{\varphi_1}$ and $b_k^{\varphi_1}$ are false: We don't need to satisfy any additional constraints.

Note that the case where there are no switching points in J_2 are covered by the Case $j < k$. The following encodes the above conditions:

$$\bigwedge_{1\leq j<k} [(t_j < \mathbf{U} \leq t_{j+1}) \Rightarrow (P^{\varphi_1}_{j+1} - (t_{j+1} - \mathbf{U}) \leq \mathbf{u} - \mathbf{l})]. \tag{19}$$

$$(t_k \in J_2 \wedge a^{\varphi_1}_k \wedge b^{\varphi_1}_k) \Rightarrow [(P^{\varphi_1}_k \leq (t_k - \mathbf{L}))\wedge (P^{\varphi_1}_k + (\mathbf{U} - t_k) \leq (\mathbf{u} - \mathbf{l}))] \tag{20}$$

$$(t_k \in J_2 \wedge \neg a^{\varphi_1}_k \wedge b^{\varphi_1}_k) \Rightarrow (\mathbf{U} - t_k \leq \mathbf{u} - \mathbf{l}) \tag{21}$$

To summarize, if $J_1 = [\mathbf{l}, \mathbf{u}]$ for some $\mathbf{l}, \mathbf{u} \in \mathbb{Z}$ and J_1 is contained in J_2, an open interval, then $\mathbb{D}(J_1, J_2)$ is the conjunction of the formulas (12), (13), (14), (15), (16), (17), (18), (19), (20), and (21). This concludes the construction of *Enc*(φ, k).

B. Correctness of the Encoding

Next, we provide the lemmas and theorems that capture the correctness of the encoding. Let us start by defining some notation.

A *valuation* $\mathbf{v}$ for *Var*(φ, k) assigns a value to each of the variables of *Var*(φ, k) of the appropriate type, that is, it assigns non-negative real numbers to each t_i, real values to each $a^{\mathbf{x}}_i$, and P^{ψ}_i, and boolean values to each a^{ψ}_i and b^{ψ}_i, and C^{ψ}_i. Given a valuation $\mathbf{v}$ for *Var*(φ, k) and a subformula ψ of φ, we denote by $\tau^{\mathbf{v},\psi}$ the boolean timed trace encoded by variables corresponding to ψ, that is, $\tau^{\mathbf{v},\psi} = (\mathbf{v}(a^{\psi}_1), \mathbf{v}(t_1), \mathbf{v}(b^{\psi}_1))(\mathbf{v}(a^{\psi}_2), \mathbf{v}(t_2), \mathbf{v}(b^{\psi}_2)) \cdots (\mathbf{v}(a^{\psi}_k), \mathbf{v}(t_k), \mathbf{v}(b^{\psi}_k))$. Similarly, $\tau^{\mathbf{v},\mathbf{x}}$ denotes the real-valued timed trace, $\tau^{\mathbf{v},\mathbf{x}} = (\mathbf{v}(a^{\mathbf{x}}_1), \mathbf{v}(t_1))(\mathbf{v}(a^{\mathbf{x}}_2), \mathbf{v}(t_2)) \cdots (\mathbf{v}(a^{\mathbf{x}}_k), \mathbf{v}(t_k))$.

The following lemma captures the relation between the real timed trace defined by the variables $a^{\mathbf{x}}_i$, and the boolean timed trace defined by the variables a^{ψ}_i and b^{ψ}_i. The proof of the lemma is by induction on the structure of φ and follows the inductive definition of *Enc*(φ, k).

Lemma 4: Let φ be an *STL* formula over variables $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{v}$ be a valuation for *Var*(φ, k) that satisfies *Form*(φ, k). Then $(pwl(\tau^{\mathbf{v},\mathbf{x}}))^{\psi} = pwc(\tau^{\mathbf{v},\psi})$.

The soundness of the encoding is summarized in the following theorem, which follows from Lemma 4. A satisfying assignment for *Enc*(φ, k) such that $a^{\varphi}_1 = \top$ corresponds to a satisfying piecewise linear signal for φ.

Theorem 1: Let φ be an *STL* formula over variables $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{v}$ be a valuation for *Var*(φ, k). If $\mathbf{v}$ satisfies the formula $a^{\varphi}_1 \wedge$ *Enc*(φ, k), then $pwl(\tau^{\mathbf{v},\mathbf{x}}) \models \varphi$.

The next theorem provides a completeness result. It states that if there is a piecewise linear signal satisfying φ, then for some k, $a^{\varphi}_1 \wedge$ *Enc*(φ, k) will return a satisfying assignment. This follows from Lemma 4 along with Lemma 3, which states that every piecewise linear signal is k-varying with respect to φ.

Theorem 2: Let φ be an *STL* formula over variables $\mathbf{x} = (x_1, \ldots, x_n)$. If there exists a piecewise linear signal $\sigma : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$ satisfying the formula φ, then $a^{\varphi}_1 \wedge$ *Enc*(φ, k) is true for some k.

C. Generating satisfying and violating traces

The formula *Enc*(φ, k) encodes all k-varying signals σ with respect to the STL formula φ. In particular, the variables a^{φ}_i and b^{φ}_i capture the signal σ^{φ}, that is, the information regarding the satisfaction of φ along the signal σ. Hence, a satisfying signal can be obtained by adding the constraint $a^{\varphi}_1 = \top$, while a violating signal can be obtained by adding the constraint $a^{\varphi}_1 = \bot$.

Further, multiple satisfying and/or violating traces can be obtained by adding blocking conditions. More precisely, one can add constraints which specify that all a^{p}_i and b^{p}_i values, for predicates p, are not the equal to their values returned in the previous satisfying assignment. This would provide a qualitatively different signal, which corresponds to a signal whose satisfaction of the predicates differs from those of the previous signals. Instead, the blocking condition could merely eliminate the particular signal by adding constraints that specify that all $a^{\mathbf{x}}_i$ and $b^{\mathbf{x}}_i$ values, for variables $\mathbf{x}$, are not the equal to their values returned in the previous satisfying assignment.

VI. Evaluation

We present a collection of results that demonstrates the value and the performance of our trace generation approach for *STL* formulae. First, we define a collection of *STL* specifications that we will use for the evaluation. The specifications are taken from the ST-Lib library of specifications, which is used to create specifications for CPS applications [16]. Next, we illustrate the value of our approach with an example that demonstrates how the sample traces can be used to debug *STL* specifications. The example shows that the traces can provide insight into issues with a specification that constrain behaviors in a way that is not intended by the designer. Lastly, we provide performance results for the method in the form of computation times.

A. Example Specifications

To evaluate the performance of our approach, we use a set of *STL* requirements drawn from ST-Lib, which is a collection of specifications that captures behaviors commonly of interest in CPS control applications [16]. The ST-Lib incorporates specifications that capture aspects of system behaviors such as overshoot, settling time, rise time, and steady-state errors. We adapt a subset of the specifications from [16]. In that work, the specifications were provided in terms of undesirable behaviors; that is, the specifications provided are satisfied when a bad behavior occurs. We use the negation of those specifications, which describes desired system behaviors.

Most of the specifications we use are defined in terms of a three classes of step functions (or lack thereof). The *STL* formula that define steps up, steps down, and steps are as follows:

$$\begin{aligned} \texttt{Step\_Up} &:= in(t+dt) - in(t) > Step\_Thresh, \\ \texttt{Step\_Down} &:= in(t) - in(t+dt) > Step\_Thresh, \\ \texttt{Step} &:= \texttt{Step\_Up} \vee \texttt{Step\_Down}, \end{aligned}$$

where dt is a constant that defines a *small* amount of time over which to test how far a signal has changed, and $Step\_Thresh$ is the constant that determines the threshold that determines whether the change over that amount of time was sufficient to classify the signal behavior as a step. For our examples, we use $dt = Step\_Thresh = 0.1$.

The following are the specifications that we will use for our evaluations.

- **Value Too High**: This simple specification constrains the maximum value of the output signal.

$$\texttt{Value\_High} \;:=\; \square_{[0,10]}(out(t) < 10)$$

- **Steady State Error High**: This specification requires that at every instant t such that there has not been a step in the input signal for SST seconds, the relative error must be lower than some value.

$$\texttt{SS\_Error\_High} \;:=\; \square_{[0,10]}(\square_{[0,SST]}(\neg\texttt{Step}) \implies \square_{[SST,SST]}(\texttt{SS\_Error\_Under\_Limit})),$$

  where SST is a time over which the system is expected to have reached a steady-state condition after some transient behavior, and

$$\texttt{SS\_Error\_Under\_Limit} \;:=\; out(t) < 1.05 \cdot in(t).$$

- **Overshoot**: This specification captures the maximum error limit allowed for a fixed time after a step up in the input signal.

$$\texttt{Overshoot} \;:=\; \square_{[0,10]}(\texttt{Step\_Up} \implies \square_{[0,SST]}(\texttt{Overshoot\_Limit})),$$

  where

$$\texttt{Overshoot\_Limit} \;:=\; out(t) < 1.1 \cdot in(t).$$

- **Rise-time**: This specification enforces that the output signal should come sufficiently close to the input signal within a given amount of time after a step up in the input value has occurred.

$$\texttt{Rise\_Time} \;:=\; \square_{[0,10]}(\texttt{Step\_Up} \implies \Diamond_{[dt,RT]}(\texttt{Over\_Rise\_Limit})),$$

  where RT is a constant that determines by which time after a step the signal should satisfy `Over_Rise_Limit`, which is define as follows:

$$\texttt{Over\_Rise\_Limit} \;:=\; out(t) > 0.9 \cdot in(t).$$

  For our experiments, we set $\texttt{Over\_Rise\_Limit} = 1.0$.

The above specifications represent constraints on the *upper* bound for error and for steps up (as opposed to steps down). Analogous specifications could be used to consider lower bounds on error and steps down, but we do not explicitly consider them here for our evaluations.

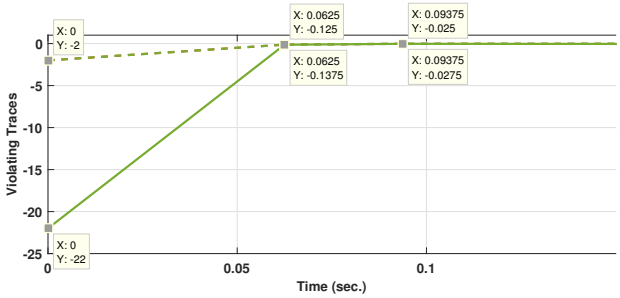

Fig. 3. Detailed view of example that demonstrates shortcoming of the `Overshoot` requirement.

B. Example Use Case

We present one example of a use case for the trace generation method presented above. Consider a designer that wishes to create a specification that constrains the upper bound on the error after a step up in the input signal. Consider further that the designer decides to use the `Overshoot` specification for this purposes and wants to evaluate whether this specification is appropriate. Using our trace generation technique, we can provide the designer with one set of trace examples that satisfy and one set of traces that do not satisfy this specification, which they can then use to check against their intention. We will show how to use the trace generation method to elucidate a shortcoming of this specification.

We use our trace generation method to compute 5 good and 5 bad examples for the `Overshoot` specification. A detailed view of the output trace of one of the resulting bad examples is showing in Fig. 3. In the figure, the dashed green line represents in, and the solid green line represents out. For this example, the output signal is always less than the input signal, and yet the signal *does not* satisfy the specification. This is because the input signal is always negative; that is, `Overshoot_Limit` can fail even though $out(t) < in(t)$ (e.g., for $in(t) = -1.0$, $out(t) = -1.05$ is lower but does not satisfy `Overshoot_Limit`). This highlights the fact that the `Overshoot` specification, along with all of the other specifications given in Sec. VI-A, assume that the input signals are always positive; otherwise, we would account for the possibility of a negative input signal by defining `Overshoot_Limit` as $out(t) < in(t) + 0.1 \cdot |in(t)|$. If the designer expected that negative input values were possible, similar adjustments would be made to `SS_Error_Under_Limit` and `Over_Rise_Limit`.

This example demonstrates how our trace generation method can be used to aid a designer during a specification development process to identify a shortcoming of a specification. In the next section, we present results from the evaluation of the computational performance of our approach, using the specifications presented in Sec. VI-A.

C. Performance Evaluation

This section provides an evaluation of the computational results of our trace generation approach. We evaluate the approach across several dimensions, including type of spec-

TABLE I
EVALUATION RESULTS FOR STL-TO-SMT TOOL.

Specification	k	No. Examples Returned								
		1	5	10	15	20	25	30	35	40
Value_High	1	2.484	2.032	3.472	3.130	2.313	2.833	3.449	3.965	4.543
	4	0.410	1.009	1.782	2.576	3.367	4.300	5.016	5.873	6.733
	8	0.506	1.472	2.695	3.975	5.257	6.553	7.847	9.416	10.549
	12	0.641	2.107	4.009	5.937	7.875	9.829	11.789	13.915	15.742
	16	0.795	2.810	5.360	7.924	10.528	13.126	15.825	18.691	21.345
	20	1.018	3.866	7.425	11.049	14.791	18.237	21.955	25.652	29.546
	24	1.271	4.952	9.797	14.358	19.244	24.115	28.842	33.796	38.862
	28	1.753	6.362	12.387	18.575	24.812	30.998	37.519	43.774	50.132
	32	2.176	9.082	17.422	25.726	33.953	42.110	50.245	58.239	66.247
SS_Error_High	1	1.174	1.888	2.622	3.430	4.257	5.088	6.036	6.875	7.690
	4	2.169	6.481	11.778	17.136	22.744	28.610	34.394	39.753	44.429
Overshoot	1	-	-	-	-	-	-	-	-	-
	4	0.961	2.498	4.462	6.482	8.517	10.699	12.867	15.121	17.251
Rise_Time	1	-	-	-	-	-	-	-	-	-
	4	0.906	2.338	4.050	5.865	7.703	9.626	11.535	13.537	15.517

ification, degree to which the signal is varying (i.e., k for a k-varying signal), and number of examples returned.

Table I presents computation time results. All times are measured in seconds. The specification name appears in the first column, and the second column indicates the number of switching instants allowed in the solutions (i.e., the k value). Each of the remaining columns correspond to a unique number of examples of satisfying and violating traces obtained using our method (e.g., the column labeled "5" corresponds to cases where an attempt is made to find 5 satisfying and 5 violating traces). Therefore each table entry corresponds to an experiment wherein a given number of satisfying and violating traces are obtained. For any experiment, if our method fails to find all of the indicated satisfying and violating traces, the corresponding table entry is populated with a dash ("-").

Note that the k values for the last three specifications are limited to 1 and 4, as larger k values result in excessive SMT solver computation times (essentially leading to a *time out* condition). This suggests poor scalability for these specifications as compared to the `Value_High` specification; this is likely due to the fact that the number of variables generated by the encoding for these specifications is larger than the number used for the `Value_High` specification. Also note that the $k = 1$ rows for the `Overshoot` and `Rise_Time` specifications are populated with dashes. This is because the violating traces for these specifications cannot be found when the signals are limited to $k = 1$, due to the fact that at least $k = 3$ is needed to generate the step behaviors necessary to demonstrate a violating behavior.

These results show that our approach can be used to generate a multiplicity of examples for a collection of STL specifications intended for industrial application. In the future, further experiments will be performed on a larger set of industrial STL specifications.

VII. CONCLUSIONS AND FUTURE WORK

We presented a method to automatically generate example traces from a formal specification language. The resulting traces help the user to better understand the behaviors permitted by and rejected by a given specification. The method uses a novel encoding from a formula given in signal temporal logic (STL) to a formula in first order logic that can be processed by a satisfiability modulo theories (SMT) solver. SMT queries that result in valuations of the query variables correspond to output traces that satisfy the STL formula. We provide proof that satisfying variable assignments correspond to traces that satisfy the STL specification. The technique can be applied to obtain examples of traces that violate the STL specification as well as examples that satisfy the specification. Using an iterative approach, we can obtain a collection of satisfying and violating traces. We demonstrated the application of our method using an implementation in MATLAB, which uses the Breach STL toolbox and the Z3 SMT solver, and we provided an evaluation of the performance of the method along with an example demonstrating how the method can be used to debug an STL specification.

A limitation of the approach is that the SMT solver exhibits poor performance for some examples that require many variables in the encoding. In future work, we will investigate more efficient encoding of STL formula into first order logic that are able to achieve similar output signals using fewer variables, which will result in improved performance. Also, in the future, further experiments will be performed on a larger set of industrial STL specifications, especially to evaluate the relative benefits of the iterative algorithm for generating a sequence of satisfying/violating traces.

REFERENCES

[1] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, 2009.

[2] Michael Bauland, Thomas Schneider, Henning Schnoor, Ilka Schnoor, and Heribert Vollmer. The complexity of generalized satisfiability for linear temporal logic. *Journal on Logical Methods in Computer Sciences*, 2009.

[3] Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. A tool for deciding the satisfiability of continuous-time metric temporal logic. *Journal on Acta Informatica*, 2016.

[4] Marcello Maria Bersani, Matteo Rossi, and Pierluigi San Pietro. Deciding the satisfiability of MITL specifications. In *International Symposium on Games, Automata, Logics, and Formal Verification*, 2013.

[5] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. Mightyl: A compositional translation from MITL to timed automata. In *International Conference on Computer Aided Verification*, 2017.

[6] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust*, 2014.
[7] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.
[8] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. *International Journal on Formal Methods in System Design*, 2017.
[9] Alexandre Donzé. On signal temporal logic. In *International Conference on Runtime Verification*, 2013.
[10] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *International Conference on Computer Aided Verification*, 2013.
[11] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In *International Symposium on Automated Technology for Verification and Analysis*, 2012.
[12] Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *International Journal on Software Tools for Technology Transfer*, 2007.
[13] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Journal on Theoretical Computer Science*, 2009.
[14] Carlo A. Furia and Paola Spoletini. Bounded variability of metric temporal logic. *Journal on Annals of Mathematics and Artificial Intelligence*, 2017.
[15] Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L. Sangiovanni Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia. Diagnosis and repair for synthesis from signal temporal logic specifications. In *International Conference on Hybrid Systems: Computation and Control*, 2016.
[16] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia. St-lib: A library for specifying and classifying model behaviors. In *The Society of Automobile Engineers Technical Papers*, 2004.
[17] Ron Koymans. Specifying real-time properties with metric temporal logic. *Journal on Real-Time Systems*, 1990.
[18] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. Ltlf satisfiability checking. In *European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems*, 2014.
[19] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. Hyperproperties of real-valued signals. In *International Conference on Formal Methods and Models for System Design*, 2017.
[20] Pavithra Prabhakar and Deepak D'Souza. On the expressiveness of MTL with past operators. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 2006.
[21] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In *International Conference on Hybrid Systems: Computation and Control*, 2015.
[22] Hendrik Roehm, Thomas Heinz, and Eva Charlotte Mayer. Stlinspector: STL validation with guarantees. In *International Conference on Computer Aided Verification*, 2017.
[23] Hendrik Roehm, Jens Oehlerking, Thomas Heinz, and Matthias Althoff. STL model checking of continuous and hybrid systems. In *International Symposium on Automated Technology for Verification and Analysis*, 2016.
[24] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Optimization-based trajectory generation with linear temporal logic specifications. In *International Conference on Robotics and Automation*, 2014.

APPENDIX

a) Proof of Lemma 2: **Base case:** If φ is a linear predicate $p(\mathbf{x})$, then any linear signal will "cross" the predicate at most once. Hence, for each segment of piecewise linear signal σ, the number of times the satisfaction of φ changes is at most once. Since, each interval of σ corresponding to a linear segment will be split into finitely many intervals with constant values in σ^{φ}, we obtain that σ^{φ} is finitely varying.

Induction step: We need to show inductively that the statement is true for the formulas $\neg\varphi, \varphi_1 \vee \varphi_2$ and $\varphi_1 U_I \varphi_2$ assuming that it holds for φ, φ_1 and φ_2. Instead of showing for the $\varphi_1 U_I \varphi_2$ case, we will show that it holds for $\varphi_1 U \varphi_2$ and $\Diamond_I \varphi_2$.

- **Case** $\neg\varphi$ From induction hypothesis, σ^{φ} is finitely varying. $\sigma^{\neg\varphi}$ is a pointwise negation of σ^{φ}. Therefore, $\sigma^{\neg\varphi}$ is finitely varying.
- **Case** $\varphi_1 \vee \varphi_2$ From induction hypotheis, σ^{φ_1}, σ^{φ_2} are finitely varying. $\sigma^{\varphi_1 \vee \varphi_2}$ at time t is the disjunction of σ^{φ_1}, σ^{φ_2} at time t. This implies that the number of switching in the $\sigma^{\varphi_1 \vee \varphi_2}$ is at most sum of the number of switching in signals σ^{φ_1} and σ^{φ_2}. Thus, $\sigma^{\varphi_1 \vee \varphi_2}$ is finitely varying.
- **Case** $\varphi_1 U \varphi_2$ We need to show that $\sigma^{\varphi_1 U_I \varphi_2}$ is finitely varying. From induction hypothesis, we know that σ^{φ_1} and σ^{φ_2} are finitely varying. Let $0 = t_0 < t_1 < \ldots < t_k < t_{k+1} = \infty$ capture all the discontinuities of σ^{φ_1} and σ^{φ_2}. We will show that $\sigma^{\varphi_1 U \varphi_2}$ remains constant in the interval (t_i, t_{i+1}) for every $0 \leq i \leq k$. Suppose $t \in (t_i, t_{i+1})$. Let γ_t^{φ} denote the sequence of truth values of φ in the intervals $(t_i, t_{i+1}), [t_{i+1}, t_{i+1}], (t_{i+1}, t_{i+2}), \ldots, (t_k, t_{k+1})$. Note that since the until U is untimed, the value of $\sigma^{\varphi_1 U \varphi_2}(t)$ is solely determined by the value of γ_t^{φ}. Since, γ_t^{φ} is the same for every $t \in (t_i, t_{i+1})$, we obtain that $\sigma^{\varphi_1 U \varphi_2}$ has the same value in the interval (t_i, t_{i+1}). Hence, $\sigma^{\varphi_1 U \varphi_2}$ is finitely varying.
- **Case** $\Diamond_I \varphi$ We need to show that $\sigma^{\Diamond_I \varphi}$ is finitely varying. From the induction hypothesis σ^{φ} is finitely varying. Let $0 = t_0 < t_1 < \ldots < t_k < t_{k+1} = \infty$ capture all the discontinuities of σ^{φ}. Let $\gamma_{t,I}^{\varphi}$ denote the sequence of truth values of φ in the intervals $(t_j, t_{j+1}), [t_{j+1}, t_{j+1}], (t_{j+1}, t_{j+2}), \ldots, (t_l, t_{l+1})$, which consist of all the intervals that intersect with the interval $t + I$. Note that $\gamma_{t,I}^{\varphi}$ determines the truth value of $\sigma^{\Diamond_I \varphi}$ at t. Also, note that $\gamma_{t,I}^{\varphi}$ need not be the same for all times $t \in (t_i, t_{i+1})$ because different $t + I$ could intersect with different intervals. However, as t moves from t_i towards t_{i+1}, the intervals that it intersects with changes only finitely many time, and hence, $\gamma_{t,I}^{\varphi}$ also changes only finitely many times. Therefore, $\sigma^{\Diamond_I \varphi}$ changes only finitely many times in any interval (t_i, t_{i+1}), and hence, is finitely varying.