Efficient Parametric Identification for STL

ABSTRACT

We describe a new algorithm for the parametric identification problem for signal temporal logic (STL), stated as follows: Given a dense-time real-valued signal w and a parameterized temporal logic formula φ, compute the subset of the parameter space that renders the formula satisfied by the signal. Unlike previous solutions, which were based on search in the parameter space or quantifier elimination, our procedure works recursively on φ and computes the evolution over time of the set of valid parameter assignments. This procedure is similar to that of monitoring or computing the robustness of φ relative to w. Our implementation and experiments demonstrate this approach can work well in practice.

ACM Reference Format:

. 2018. Efficient Parametric Identification for STL. In *Proceedings of ACM International Conference on Hybrid Systems: Computation and Control (HSCC'18).* ACM, New York, NY, USA, 10 pages. https://doi.org/

1 INTRODUCTION

Signal temporal logic (STL) [33, 35] is an extension of temporal logic to handle real-valued dense-time signals which gained a lot of popularity in recent years as a rigorous and expressive formalism to describe behaviors of continuous and hybrid systems in various domains such as analog circuits [28], systems and synthetic biology [8, 17, 40], biomedical systems [13, 14] and cyber-physical control systems [15, 19, 30, 36, 37]. The reader is referred to [9] for an introduction and a survey of applications.

The major use of STL is in monitoring: a signal w, which is the output of a simulator or a sequence of measurements from a real system, is observed, and a monitoring procedure then checks whether it satisfies an STL formula φ, a fact denoted as $w \models \varphi$. The procedure for checking satisfaction works along two dimensions, one is related to the parse tree of the formula and one to time. For each sub-formula φ' a *satisfaction signal* is computed whose value at t indicates whether w satisfies φ' from t. The computation of satisfaction of a formula like $\Diamond_{[a,b]}\, \varphi$ (eventually φ within $r \in [a, b]$ time) at t is propagated backwards from the satisfaction of φ at the interval $[t + a, t + b]$.

The inverse problem of parametric identification for STL has been introduced in [6]. It uses PSTL, a parametric version of STL, admitting formulas where some of the constants in the numerical predicates and quantitative timing operators are replaced by parameters taken from $P = \{p_1, \dots, p_k\}$ ranging over some parameter space $V \subseteq \mathbb{R}^k$. Each selection of parameter values $v \in V$ transforms a PSTL formula φ into an STL formula $\varphi[v]$ that might be satisfied or not by a given signal w. The problem solved in [6] is to compute the *validity domain* of φ relative to w, that is, the set of parameter valuations v such that $w \models \varphi[v]$. This result provides an enabling technology for applying learning and data-mining techniques to observed behaviors of cyber-physical systems [10, 23, 25–27, 39, 41].

HSCC'18, April 2018, Porto, Portugal

The major result of [6] states that when signals, presented as a sequence of time-stamped values, are interpreted as piecewise-linear, the validity domains are semi-linear sets. Two approaches were proposed to compute them. The first was based on quantifier elimination in linear arithmetics, for a formula whose size is linear in the number of sampling points of the signal. The other approach was approximate, based on conducting search over the parameter space. Under certain reasonable assumptions of monotonicity in parameter influence, the latter problem is equivalent to approximating the Pareto front in multi-criteria optimization, e.g. [31].

In this paper we propose an alternative approach which resembles the way monitoring is done for qualitative (satisfaction) and quantitative (robustness) semantics. Robustness was defined and computed for STL in [18, 20] following [21, 22] and [38] and its relation to parametric identification is worth discussing. The robustness $\rho(\varphi, w)$ is a real number which is positive if and only if w satisfies φ. Moreover, all signals whose pointwise distance from w is less than $\rho(\varphi, w)$ have the same satisfaction status as w.

Consider the constraint $x \geq 0$ satisfied by some signal w in which variable x has value c. The robustness of this satisfaction is defined as $\rho(x \geq 0, w) = c$. On the other hand, the validity domain of the PSTL formula $x \geq p$ is $D(x \geq p, w) = \{v : v \leq c\}$. Consider now the formula $x \geq 0 \vee y \geq 0$ which depends on two variables. According the the common definition of robustness we will have

$$\rho(x \geq 0 \vee y \geq 0, w) = \max\{\rho(x \geq 0, w), \rho(y \geq 0, w)\}$$

a one-dimensional object that mixes the tolerances associated with the two variables. On the other hand, the validity domain for the PSTL formula $x \geq p \vee y \geq q$ is a two-dimensional object providing more refined information concerning the possible deformations of the signal that do not change satisfiability status.

In signal temporal logic, the above quantities are not constants but signals themselves. The major contribution of the paper is a new procedure to compute validity domains that follows the computation of satisfaction [33] and robustness [18] signals by propagating them as function of time, from sub-formulas to formulas. With each formula φ and a signal w we associate a parametric validity signal whose value at t indicates the set of parameter valuations v such that $(w, t) \models \varphi[v]$. The crucial component is to compute the validity domain of $\Diamond_{[a,b]}\, \varphi$ from that of φ. Like the robustness computation in [18], this involves aggregating values of a signal over a shifting window. However, we are dealing with a multi-dimensional partially-ordered parameter space, and the validity domains are typically Pareto-like sets.

Technically, we consider PSTL formula with space parameters and signals which are interpreted using a piecewise-constant interpolation. In this special case of [6], we show that time can be partitioned into finitely many intervals and the validity domain in each interval is a finite union of rectangles. We implemented the procedure for computing validity signals and demonstrate its performance on rather long signals. The extension to piecewise-linear signals and to timing parameters is discussed at the end of the paper.

2 PARAMETRIC SIGNAL TEMPORAL LOGIC

Parametric Signal Temporal Logic (PSTL) [6] is an extension of the logic STL introduced in [33] with parameters. STL enables specifying properties of Boolean and real-valued signals, through atomic formulas of the form $x \geq c$, and temporal formulas of the form $\Diamond_{[a,b]}\,\varphi$. Additionally, the logic provides Boolean connectives, and the temporal *until* operator. For the sake of simplicity we omit Boolean variables from the syntax, and only consider the case of closed timing intervals. Formula $x \geq c$ is satisfied at time instants where x is above c, while formula $\Diamond_{[a,b]}\,\varphi$ is satisfied at time instants where subformula φ holds within a to b time units in the future. Here a, b, c are constant timing and space values. The parametrization considered in PSTL enables these to be undetermined, real-valued parameters. In this work, we only consider space parameters, i.e. values a, b are constants.

A signal w is a function $\mathbb{T} \to \mathbb{R}^n$ where $\mathbb{T} = [0, d]$ is a subset of $\mathbb{R}_{\geq 0}$ which we call *time domain*. The value d is the *duration* of the signal, and we denote it by $|w|$. The value of signal w at time $t \in \mathbb{T}$ is denoted $w[t] \in \mathbb{R}^n$. Signal values are accessed by variables from the set $X = \{x_1, \ldots, x_n\}$. The projection of w onto some variable $x \in X$ is denoted w_x. Using these conventions, $w_x[t]$ denotes the value of variable x at time t given in signal w.

Let $P = \{p_1, \ldots, p_k\}$ be the set of parameters. A *parameter valuation* is a vector v that assigns a value to every parameter. We assume that parameter valuations range over a *parameter space* $V \subseteq \mathbb{R}^k$. The value of parameter a $p \in P$ in a valuation $v \in V$ is denoted $v_p \in \mathbb{R}$. We often use logic notation to describe sets of parameter valuations. For example, we write $p_1 \geq 1 \wedge p_2 \leq 2$ to denote the set of parameter valuations $\{v \in V \mid v_{p_1} \geq 1 \wedge v_{p_2} \leq 2\}$.

Definition 2.1 (PSTL Syntax). Formulas φ of PSTL are described by the following grammar:

$$\begin{gathered}\varphi ::= \text{true} \mid x \leq c \mid x \geq c \mid x \leq p \mid x \geq p \mid \\ \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{[a,b]}\,\varphi \mid \varphi_1\,\mathcal{U}\,\varphi_2\end{gathered}$$

where $x \in X$ is a signal variable, $c \in \mathbb{R}$ is a constant, $p \in P$ is a parameters, and $0 \leq a \leq b \in \mathbb{R}$ are positive constants,

We also use standard abbreviations $x < c \equiv \neg(x \geq c)$, $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, and common temporal operators:

$$\Diamond\,\varphi \equiv \text{true}\,\mathcal{U}\,\varphi \qquad \Box\,\varphi \equiv \neg\Diamond\neg\varphi \qquad \Box_{[a,b]}\,\varphi \equiv \neg\Diamond_{[a,b]}\neg\varphi$$

The *timed until* operators $\mathcal{U}_{[a,b]}$ and $\mathcal{U}_{[a,\infty)}$ are often considered primitive, but can also be derived from the *untimed until* and *bounded eventually* as follows [35]:

$$\begin{aligned}\varphi\,\mathcal{U}_{[a,\infty)}\,\psi &\equiv \Box_{[0,a]}(\varphi\,\mathcal{U}\,\psi)\\ \varphi\,\mathcal{U}_{[a,b]}\,\psi &\equiv (\Diamond_{[a,b]}\,\psi) \wedge (\varphi\,\mathcal{U}_{[a,\infty)}\,\psi)\end{aligned}$$

Finally, we introduce the *release* operator $\mathcal{R}$, as the dual of until:

$$\varphi\,\mathcal{R}\,\psi \equiv \neg(\neg\varphi\,\mathcal{U}\,\neg\psi)$$

STL can be defined as the subset of PSTL formulas free of parameters. A parameter valuation $v \in V$ transforms a PSTL formula φ into the STL formula denoted $\varphi[v]$ obtained by replacing in φ every parameter p with its value $v_p \in \mathbb{R}$.

We now recall the semantics of STL from [33].

Definition 2.2 (STL Semantics). The satisfaction of STL formula φ by signal w at time $t \in \mathbb{T}$, denoted by $(w,t) \models \varphi$, is defined inductively as follows:

$$\begin{array}{lll}(w,t) \models \text{true} & & \\ (w,t) \models x \geq c & \text{iff} & w_x[t] \geq c\\ (w,t) \models x \leq c & \text{iff} & w_x[t] \leq c\\ (w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi\\ (w,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2\\ (w,t) \models \Diamond_{[a,b]}\,\varphi & \text{iff} & \exists t' \in t \oplus [a,b],\ (w,t') \models \varphi\\ (w,t) \models \varphi_1\,\mathcal{U}\,\varphi_2 & \text{iff} & \exists t' \in [t,\infty),\ (w,t') \models \varphi_2 \text{ and}\\ & & \forall t'' \in (t,t'),\ (w,t'') \models \varphi_1\end{array}$$

We say that w satisfies φ, written $w \models \varphi$, when $(w,0) \models \varphi$.

For simplicity, we use non-strict *until* semantics, meaning that ψ is allowed to hold at the current point in time, rather than strictly in future. See [24] for a discussion of strict versus non-strict semantics.

The introduction of parameters enables to consider several variants of the same formula, with different constants. This is captured in the notion of *validity domain.*

Definition 2.3 (Validity Domain). The validity domain of a PSTL formula φ relative to some signal trace w, denoted by $D(\varphi, w)$, is the set of parameter valuations for which the formula is satisfied:

$$D(\varphi, w) = \{v \in V : w \models \varphi[v]\}$$

Our method explicitly computes the validity domain of the formula in a bottom-up style of computation. Starting from the validity domain of atomic formulas, which derives directly from the input signals, the validity domain of non-atomic formulas is assembled by Boolean and temporal combinations of validity domains of its subformulas. In the case of temporal operators, we require the validity domain of the subformulas at future time points.

Definition 2.4 (Parametric Validity Signal). Given a PSTL formula φ, a signal w, the *parametric validity signal* denoted $d(\varphi, w) : \mathbb{T} \to \wp(V)$ is defined as follows:

$$d(\varphi, w)[t] = \{v \in V : (w,t) \models \varphi[v]\}$$

Lemma 2.5 (Inductive Characterization). *For a PSTL formula, a signal w, and a time point $t \in \mathbb{T}$ we have*

$$\begin{aligned}d(x \leq c, w)[t] &= \textit{if } w_x[t] \leq c \textit{ then } V \textit{ else } \varnothing\\ d(x \geq c, w)[t] &= \textit{if } w_x[t] \geq c \textit{ then } V \textit{ else } \varnothing\\ d(x \leq p, w)[t] &= \{v \in V : w_x[t] \leq v_p\}\\ d(x \geq p, w)[t] &= \{v \in V : w_x[t] \geq v_p\}\\ d(\neg\varphi, w)[t] &= V \setminus d(\varphi, w)[t]\\ d(\varphi \vee \psi, w)[t] &= d(\varphi, w)[t] \cup d(\psi, w)[t]\\ d(\Diamond_{[a,b]}\,\varphi, w)[t] &= \bigcup_{t' \in t \oplus [a,b]} d(\varphi, w)[t']\\ d(\varphi\,\mathcal{U}\,\psi, w)[t] &= \bigcup_{t' \geq t}\left(d(\psi, w)[t'] \cap \bigcap_{t'' \in (t,t')} d(\varphi, w)[t'']\right)\end{aligned}$$

where $x \in X$, $c \in \mathbb{R}$, and $p \in P$.

When the satisfaction of the formula is monotonic in each of its parameters, one may equivalently talk about the set of tightest parameter valuations such that the formula is satisfied by the signal.

We now wish to define the *polarity* of a parameter $p \in P$ in φ, with the intention to assign *positive* polarity to p if φ is easier to satisfy as the value of p increases, and *negative* polarity if φ is harder to satisfy as p increases. Obviously, formula satisfaction may not be monotonic in p, and p may not have a defined polarity.

Definition 2.6. The polarity $\pi(p, \varphi)$ of a parameter $p \in P$ in a formula φ is defined by induction as follows:

$$\begin{aligned}
\pi(\text{true}, p) = \pi(x \le c, p) = \pi(x \ge c, p) &= \{-1, 1\} \\
\pi(x \le p', p) &= \{1\} \text{ if } p' = p, \text{ otherwise } \{-1, 1\} \\
\pi(x \ge p', p) &= \{-1\} \text{ if } p' = p, \text{ otherwise } \{-1, 1\} \\
\pi(\neg\varphi, p) &= \{-i : i \in \pi(\varphi, p)\} \\
\pi(\varphi \vee \psi, p) &= \pi(\varphi, p) \cap \pi(\psi, p) \\
\pi(\Diamond_{[a,b]}\, \varphi, p) &= \pi(\varphi, p) \\
\pi(\varphi\, \mathcal{U}\, \psi, p) &= \pi(\varphi, p) \cap \pi(\psi, p)
\end{aligned}$$

Intuitively, 1 and -1 denote positive and negative polarity respectively. The value $\varnothing$ means that different subformulas of φ give different polarity to p, and thus p does not have consistent polarity within. The value $\{-1, 1\}$ means that p does not occur in φ

In this work, we restrict ourselves to formulas whose parameters have consistent polarity, which is a common case. The satisfaction of such formulas is monotonic w.r.t. parameter values and this allows us to characterize signals in terms of the tightest parameter values with which a given formula is satisfied. Restricting to consistent polarity does not incur a loss of generality, in the following sense. Given a formula φ with an inconsistent parameter p, we can replace its negative occurrences with a fresh parameter p', and keep the positive occurrences untouched. Then we can intersect the validity domain of this new formula with the plane $p = p'$ to obtain the validity domain of the original formula.

In addition to assuming consistent polarity, we assume that the formula is given to our algorithm in negation normal form. That is, our input language is described by the following grammar:

$$\begin{aligned}
\varphi ::= {} & \text{true} \mid \text{false} \mid x \prec c \mid x \succ c \mid x \prec p \mid x \succ p \mid \\
& \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_{[a,b]}\, \varphi \mid \Box_{[a,b]}\, \varphi \mid \varphi_1\, \mathcal{U}\, \varphi_2 \mid \varphi_1\, \mathcal{R}\, \varphi_2
\end{aligned}$$

where $\prec\, \in \{<, \le\}$ is a comparison operator, and the meaning of x, c, p, a, and b is as before. Every formula φ has an equivalent in negation normal form, produced by rewriting φ using standard arithmetic and logical rules and the following temporal equivalences:

$$\neg \Diamond_{[a,b]}\, \varphi \Leftrightarrow \Box_{[a,b]}\, \neg\varphi \qquad \neg(\varphi\, \mathcal{U}\, \psi) \Leftrightarrow \neg\varphi\, \mathcal{R}\, \neg\psi$$

Finally, we restrict ourselves to computing the *topological closure* of the validity domain. One can also see this as computing the boundary of the validity domain disregarding the fact whether the boundary lies inside or outside of it. Syntactically, taking the topological closure of the domain is equivalent to replacing strict inequalities in the formula with their non-strict versions. We assume that our formulas are in negation normal form and only contain non-strict inequalities $x \le c$, $x \ge c$, $x \le p$, $x \ge p$.

3 COMPUTING THE VALIDITY SIGNALS

In this section, we study the case of piecewise-constant input signals, which we represent as finite sequences of time intervals mapped to values. We show how to compute the validity signals of formulas by induction on their structure.

Definition 3.1 (Piecewise-Constant Signal Representation). A piecewise-constant signal w is represented as a sequence of intervals $[t_{i-1}, t_i)$ mapped to values w^i:

$$\langle [t_0, t_1) \mapsto w^1; [t_1, t_2) \mapsto w^2; \cdots ; [t_{n-1}, t_n) \mapsto w^n \rangle$$

where $t_0 = 0$, $t_n = |w|$ is the duration of w, and w^i is the value of $w[t]$ when $t \in [t_{i-1}, t_i)$. We say that n is the *length* of the signal. We call *segment* a mapping of an interval to a constant value, written $[t_{i-1}, t_i) \mapsto w^i$.

For a piecewise-constant input signal, the validity signal is also piecewise-constant.

Theorem 3.2. *For a formula φ and piecewise-constant signal w,*

(1) *$d(\varphi, w)$ is a piecewise-constant signal;*
(2) *$d(\varphi, w)[t]$ is finite union of rectangles in V for all $t \in \mathbb{T}$.*

The rest of the section can be seen as a constructive proof of this.

For the Boolean constants and atomic comparisons, we can directly build the piecewise-constant validity signals, where the validity domains are rectangles. For the logical and temporal operators, we combine the validity signals of the subformulas using a finite number of set unions and intersections. We process the validity signals of the subformulas piece by piece, and concatenate the resulting pieces together. We now introduce additional notation for this purpose.

Let $w = \langle [t_0, t_1) \mapsto w^1; \cdots ; [t_{n-1}, t_n) \mapsto w^n \rangle$ be a signal and let $s = [\tau, \tau') \mapsto w'$ be a new segment. When $\tau = t_n$, we may *append* the new segment to w. We write $\langle w; s \rangle$ to denote the result of appending. When $t_0 > 0$ (i.e., w is not a fully constructed signal) and $\tau' = t_0$, we may *prepend* the new segment to w. We write $\langle s; w \rangle$ to denote the result of prepending. We write $\langle \rangle$ to denote the empty signal, that has no segments and has duration 0. The empty signal can be appended or prepended to any signal s, which leaves it unaffected, that is, $\langle \langle \rangle; s \rangle = \langle s; \langle \rangle \rangle = s$.

Overall Algorithm. The algorithm proceeds by induction on the formula structure. For Boolean constant and atomic comparisons, we compute the validity signal directly, by iterating over the segments of the input signal. For the logical and temporal operators, we first compute the full validity signals of the subformulas. Below we explain how to compute the validity signal for every type of formula. For every operator, the computation step implements the corresponding relation of Lemma 2.5.

The algorithms in this section view validity domains as plain set of parameter valuations (i.e., subsets of V) without any internal structure, and combine them with set union and set intersection. We will later discuss the properties of validity domains of consistent-polarity formulas and how these domains can be efficiently represented.

Boolean Constants. For the boolean constants, the validity signals are the constant functions:

$$d(\text{true}, w) = \langle [0, |w|) \mapsto V \rangle \qquad d(\text{false}, w) = \langle [0, |w|) \mapsto \varnothing \rangle$$

```
function Combine(d^φ, d^ψ, f)
  let d^φ = ⟨[t_{i−1}, t_i) ↦ d^φ_i⟩_{i=1..n}
  let d^ψ = ⟨[τ_{j−1}, τ_j) ↦ d^ψ_j⟩_{j=1..m}
  res ← ⟨⟩ ,  i ← j ← 1
  while i ≤ n ∧ j ≤ m do
    res ← ⟨res; [max{t_{i−1}, τ_{j−1}}, min{t_i, τ_j}) ↦ f(d^φ_i, d^ψ_j)⟩
    if t_i = min{t_i, τ_j} then i ← i + 1
    if τ_j = min{t_i, τ_j} then j ← j + 1
  return res
```

Figure 1: Combining two validity signals d^φ and d^ψ with a given function f.

Atomic Comparisons. Let the component x of the input signal w be represented as the sequence $\langle [t_{i-1}, t_i) \mapsto w_x^i \rangle_{i=1..n}$. Then, the validity domains for atomic constraints are computed in the following way. For the comparisons with constants $x \le c$ and $x \ge c$,

$$d(x \le c, w) = \langle [t_{i-1}, t_i) \mapsto \text{if } w_x^i \le c \text{ then } V \text{ else } \varnothing \rangle_{i=1..n}$$
$$d(x \ge c, w) = \langle [t_{i-1}, t_i) \mapsto \text{if } w_x^i \ge c \text{ then } V \text{ else } \varnothing \rangle_{i=1..n}$$

For the comparison with parameters $x \le p$ and $x \ge p$,

$$d(x \le p, w) = \langle [t_{i-1}, t_i) \mapsto \{v \in V : v_p \ge w_x^i\} \rangle_{i=1..n}$$
$$d(x \ge p, w) = \langle [t_{i-1}, t_i) \mapsto \{v \in V : v_p \le w_x^i\} \rangle_{i=1..n}$$

Disjunction and Conjunction. The validity signals $d(\varphi \vee \psi, w)$ and $d(\varphi \wedge \psi, w)$ are produced by combining the validity signals $d(\varphi, w)$ and $d(\psi, w)$ point-wise with set union and set intersection respectively. In Fig. 1, we show the function *Combine* that produces such a combination for two piecewise-constant validity signals of the same duration. We have

$$d(\varphi \vee \psi, w) = \mathit{Combine}(d(\varphi, w), d(\psi, w), \cup)$$
$$d(\varphi \wedge \psi, w) = \mathit{Combine}(d(\varphi, w), d(\psi, w), \cap)$$

Bounded Eventually and Always. Given a time point t, the value of the validity signal $d(\Diamond_{[a,b]}\,\varphi, w)$ at time t is the union of values of $d(\varphi, w)$ on the interval $[t + a, \min\{t + b, |w|\}]$. We call this interval the *forward cone* of t w.r.t. $[a, b]$. Conversely, we observe that given a time point t', for every $t \in [\min\{0, t' - b\}, t' - a]$ we have $d(\Diamond_{[a,b]}\,\varphi, w)[t] \subseteq d(\varphi, w)[t']$. We call the interval $[\min\{0, t' - b\}, t' - a]$ the *backward cone* of t'. This definition can be lifted to an interval $[t, t')$ for which the $[a, b]$-backward cone is defined as

$$[t, t') \ominus [a, b] = [\max\{0, t - b\}, t' - a).$$

This operation is based on Minkowski difference (taking into account that the time domain is bounded) and we denote it accordingly. Now, if the piecewise-constant representation of $d(\varphi, w)$ contains a segment $[t_{i-1}, t_i) \mapsto d_i^\varphi$ then for every $t \in [t_{i-1}, t_i) \ominus [a, b]$ we have $d(\Diamond_{[a,b]}\,\varphi, w)[t] \subseteq d_i^\varphi$. This leads us to the idea that we can compute $d(\Diamond_{[a,b]}\,\varphi,)$ by *backshifting*: for every segment in the piecewise-constant representation of $d(\varphi, w)$ we produce its backward cone and then combine the backward cones with set union.

Let us consider an example, shown in Fig. 2. At the top of the figure, we show the input validity signal that talks about a single parameter p. It maps the interval $[0, 3)$ to the set $\{v : v_p \ge 3\}$, $[3, 6)$ to $\{v : v_p \ge 2\}$, $[6, 9)$ to $\{v : v_p \ge 4\}$, $[9, 17)$ to $\{v : v_p \ge 3\}$, and $[17, 20)$ to $\{v : v_p \ge 0\}$. Below, we show the backward cones of these intervals w.r.t. the time window $[1, 8]$. For this time window, the backward cone of $[0, 3)$ is $[0, 2)$, the backward cone of $[9, 17)$ is $[1, 16)$ and so on. At the bottom of the figure we show the result of combining the backward cones with set union. Notice, how no backward cones overlap the interval $[19, 20)$ and thus the result maps this interval to the empty set.

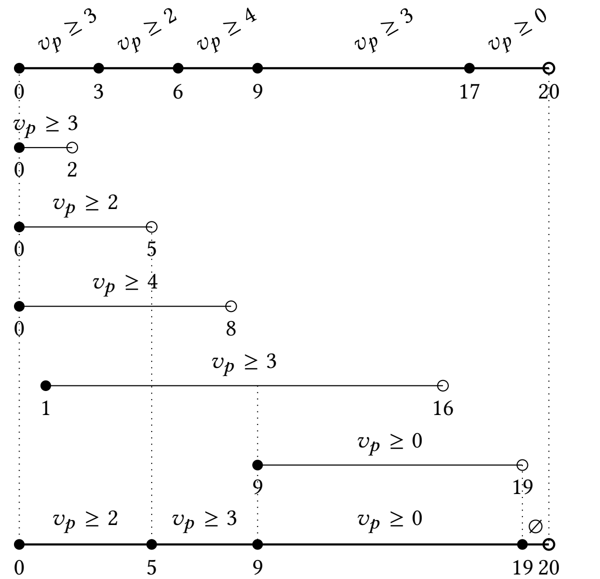

Figure 2: Example of backshifting by a time window $[1, 8]$, with $f = \cup$.

The algorithm *Backshift* in Figures 3–5 is designed to combine the backward cones efficiently. The combination is done with set union for the *eventually* operator and with set intersection for *always*. For a number of initial segments of $d(\varphi, w)$, their backward cones start at time 0 (in the example in Fig.2, these are segments $[0, 3)$, $[3, 6)$, and $[6, 9)$). We can produce the intermediate result of combining just these backward cones by scanning $d(\varphi, w)$ backwards and computing the running union or intersection. This is done by the function *BackshiftInit* in Fig. 4. The remaining segments of $d(\varphi, w)$ are combined by scanning them forward and calling the function *BackshiftAdd* (in Fig. 5) for every backward cone.

BackshiftAdd scans the intermediate result of backshifting *res* backwards and combines the validity domains of the existing segments d_i and the validity domain of the new segment d'. An important property is that in the intermediate result of backshifting *res*, segments that overlap with the incoming backward cone $[\tau, \tau')$ are always arranged in descending order when combining with set union and in ascending order when combining with intersection. This means that *BackshiftAdd* can stop scanning *res* when it encounters a validity domain which is a superset of d'. We return to this point later when discussing complexity, in Section 5.

Finally, we get

$$d(\Diamond_{[a,b]}\,\varphi, w) = \mathit{Backshift}(d(\varphi, w), a, b, \cup, \varnothing)$$
$$d(\Box_{[a,b]}\,\varphi, w) = \mathit{Backshift}(d(\varphi, w), a, b, \cap, \top)$$

```
function Backshift(d^φ, a, b, f, ⊥_f)
  let d^φ = ⟨[t_{i−1}, t_i) ↦ d_i⟩_{i=1..n}
  res ← BackshiftInit(d^φ, a, b, f, ⊥_f)
  i ← min i, s.t. t_{i−1} − b > 0
  while i ≤ n do
    BackshiftAdd(res, [t_{i−1} t_i) ⊖ [a, b], d^φ_i, f)
    i ← i + 1
  if a > 0 then res ← ⟨res; [|res|, |w|) ↦ ⊥_f⟩
  return res
```

Figure 3: Backshifting algorithm. Here d^φ is the validity signal to be backshifted; backward cones are computed w.r.t. the interval $[a, b]$; overlapping backward cones are combined with the function f; $\bot_f$ is the neutral element w.r.t. f.

```
function BackshiftInit(d^φ, a, b, f, ⊥_f)
  let d^φ = ⟨[t_{i−1}, t_i) ↦ d^φ_i⟩_{i=1..n}
  i ← max i, s.t. t_{i−1} − b ≤ 0
  res = ⟨⟩ , d_run ← ⊥_f
  while i ≥ 1 ∧ t_i − a > 0 do
    d ← f(d_run, d^φ_i)
    res ← ⟨[t_{i−1} t_i) ⊖ [a, b] ↦ d_run; res⟩
    i ← i − 1
  return res
```

Figure 4: Backshifting the initial segments, for which the backward cones start at time 0. Here d^φ is the validity signal to be backshifted; backward cones are computed w.r.t. the interval $[a, b]$; overlapping backward cones are combined with the function f; $\bot_f$ is the neutral element w.r.t. f.

```
function BackshiftAdd(res, [τ, τ′), d′, f)
  if τ′ ≤ 0 then return
  else if res = ⟨⟩ then res ← ⟨[0, τ′) ↦ d′⟩
  else
    let res = ⟨s_1, ··· , s_m⟩ and s_i = [t_{i−1}, t_i) ↦ d_i
    i ← m
    while i ≥ 1 ∧ τ < t_i do
      if f(d_i, d′) = d_i then break
      else if τ > t_{i−1} then
        replace s_i
        with ⟨[t_{i−1}, τ) ↦ d_i; [τ, t_i) ↦ f(d_i, d′)⟩
      else
        replace s_i with [t_{i−1}, t_i) ↦ f(d_i, d′)
      i ← i + 1
    res ← ⟨res; [|res|, τ′) ↦ d′⟩
```

Figure 5: Adding a backward cone to the intermediate backshifting result *res*. Here $[\tau, \tau') \mapsto d'$ is the new backward cone and its corresponding validity domain; overlapping segments are combined with the function f.

```
function Until(d^φ, d^ψ, f, g, ⊥_f)
  let d^φ = ⟨[t_{i−1}, t_i) ↦ d^φ_i⟩_{i=1..n}
  let d^ψ = ⟨[τ_{j−1}, τ_j) ↦ d^ψ_j⟩_{j=1..m}
  res ← ⟨⟩ , i ← n , j ← m , d_fut ← ⊥_f
  while true do
    res ← ⟨[max{t_{i−1}, τ_{j−1}}, min{t_i, τ_j}) ↦ f(d^ψ_j, d_fut); res⟩
    t_end ← max{t_i, τ_j}
    if i = 1 ∧ j = 1 then break
    if τ_{j−1} > t_{i−1} then
      d_fut ← f(d_fut, g(d^φ_i, d^ψ_j)) , j ← j − 1
    else if t_{i−1} > τ_{j−1} then
      d_fut ← g(d_fut, d^φ_{i−1}) , i ← i − 1
    else
      d_fut ← g(f(d^ψ_j, d_fut), d^φ_{i−1}) , i ← i − 1 , j ← j − 1
  return res
```

Figure 6: Computing the validity signal of *until* and *release*. Here d^φ and d^ψ are the validity signals of the subformulas; $(f, g, \bot_f)$ are $(\cup, \cap, \varnothing)$ for *until* and $(\cap, \cup, V)$ for *release*.

Until and Release. The computation of the validity domain of $\varphi\,\mathcal{U}\,\psi$ is better explained in the case when $d(\varphi, w)$ and $d(\psi, w)$ are represented using the same sequence of intervals. That is, $d(\varphi, w) = \langle [t_{i-1}, t_i) \mapsto d^\varphi_i \rangle_{i=1..n}$, $d(\psi, w) = \langle [t_{i-1}, t_i) \mapsto d^\psi_i \rangle_{i=1..n}$, and thus $d(\varphi\,\mathcal{U}\,\psi, w) = \langle [t_{i-1}, t_i) \mapsto d^{\mathcal{U}}_i \rangle_{i=1..n}$. Then, the validity domains $d^{\mathcal{U}}_i$ can be inductively defined as follows. To satisfy $\varphi\,\mathcal{U}\,\psi$ at time $t \in [t_{n-1}, t_n)$ during the last time interval, we have to satisfy ψ. That is, $d^{\mathcal{U}}_n = d^\psi_n$. For $1 \le i < n$, to satisfy $\varphi\,\mathcal{U}\,\psi$ at time $t \in [t_{i-1}, t_i)$, we have to satisfy ψ; or we have to satisfy $\varphi\,\mathcal{U}\,\psi$ during the following time interval $[t_i, t_{i+1})$ and also satisfy φ on $[t_{i-1}, t_i)$. That is, for $1 \le i < n$, we have $d^{\mathcal{U}}_i = d^\psi_i \cup (d^\varphi_i \cap d^{\mathcal{U}}_{i+1})$.

The function *Until* in Fig. 6 generalizes this inductive definition for the case when $d(\varphi, w)$ and $d(\psi, w)$ are represented using different sequences of intervals. It is also parameterised with two operations, f and g. For *until*, f is set union, and g is set intersection. For *release*, this is the other way around. The function scans both input validity signals backwards, maintaining a pair of pointers, i and j. The interval $[\max\{t_{i-1}, \tau_{j-1}\}, \min\{t_i, \tau_j\})$ is the *current* interval, on which both $d(\varphi, w)$ and $d(\psi, w)$ do not change. Then, the value of $d(\varphi\,\mathcal{U}\,\psi, w)$ on the current interval is the union of the corresponding value of $d(\psi, w)$ and the set d_{fut}, which is the validity domain on the future interval intersected with the current value of $d(\varphi, w)$. Thus,

$$d(\varphi\,\mathcal{U}\,\psi, w) = \mathit{Until}(d(\varphi, w), d(\psi, w), \cup, \cap, \varnothing)$$
$$d(\varphi\,\mathcal{R}\,\psi, w) = \mathit{Until}(d(\varphi, w), d(\psi, w), \cap, \cup, V)$$

Below we discuss further details and optimization used in our implementation.

Upward-Closed Rectangular Sets. Define partial order $\le$ on V in the standard way: $v \le v'$ iff $v_i \le v'_i$ for all $i = 1, \ldots, k$. The upward closure of a point v in the partially-ordered set V is the set $\{v' : v \ge v\}$. A subset of $U \subseteq V$ is upward-closed and rectangular if it is an upward closure of a finite set of points $\downarrow U$ which is called its *support set*. The support set $\downarrow U$ can be used to represent the upward-closed set U compactly. For example, the set $\{v \in \mathbb{R}^2 : (v_1 \geq 0 \wedge v_2 \geq 1) \vee (v_1 \geq 1 \wedge v_2 \geq 0)\}$ is the upward closure of the pair of points $\{(0,1),(1,0)\}$. Set-theoretic operations (union, intersection, etc) on upward-closed rectangular sets can usually be translated to operations on their support sets. Efficient implementation of set union is usually studied in the context of maintaining a Pareto front in multi-objective optimization. For that, support sets are usually stored in a tree-like structure or, for dimensions up to 2, in a sorted list. In our implementation, we store support sets as unsorted arrays and leave the use of more efficient data structures for future work.

For the formulas with consistent polarity, we can actually make all the validity domains upward-closed, if for a negative parameter p, we interpret v_p as the opposite of the value of p. This is equivalent to replacing in a formula every negative parameter p with a positive parameter $-p$ (as done in [6]). This allows to represent validity domains as sets of their support points and also to restate Theorem 3.2 to talk about finite unions of *upward-closed* rectangles.

Distributing Temporal Operators over Boolean. Before running the computation, by default, we rewrite the input formula by distributing *eventually* over disjunction and *always* over conjunction, using the equivalences:

$$\Diamond_{[a,b]}(\varphi \vee \psi) \Leftrightarrow \Diamond_{[a,b]}\,\varphi \vee \Diamond_{[a,b]}\,\psi$$
$$\Box_{[a,b]}(\varphi \wedge \psi) \Leftrightarrow \Box_{[a,b]}\,\varphi \wedge \Box_{[a,b]}\,\psi$$

It often the case that φ and ψ have different sets of parameters, thus $d(\varphi, w)$ and $d(\psi, w)$ take values of smaller size and dimension than $d(\varphi \vee \psi, w)$ and usually have shorter sequences of incomparable values (we discuss this further in Section 5). In some of our experiments, this rewriting reduced the runtime of backshifting up to a factor of 10.

Merging Adjacent Segments. In our implementation of the algorithms (*Combine*, *Backshift*, *Until*), we maintain the invariant that in the representation of a piecewise-constant signal there are no two adjacent intervals that map to the same value. For clarity of presentation though, we do not show this in the pseudocode of the algorithms. Maintaining this invariant does not come for free. Sometimes, we can rely on the properties of underlying operations (e.g., the implementation of set union can detect the cases when the result is equal to one of the operands), but sometimes, we have to perform extra set comparisons explicitly. Maintaining this invariant may greatly reduce the length of validity signals for *until* and *eventually/always* over wide temporal windows. In our experiments, applying a temporal operator to a signal with tens of thousands of segments, could produce the validity signal that has just about a hundred of segments.

4 EXAMPLES

In this section, we illustrate the outcome of the algorithm as applied to the example signal that we show in Fig. 7. Its components, x and y, are square waves that alternate between two stable values: 0 and 2, but for a short period of time (for 1 unit) can take the transient value 1. They can be viewed as coarse quantizations of some periodic signals and the reader can imagine how they can be refined in space and time.

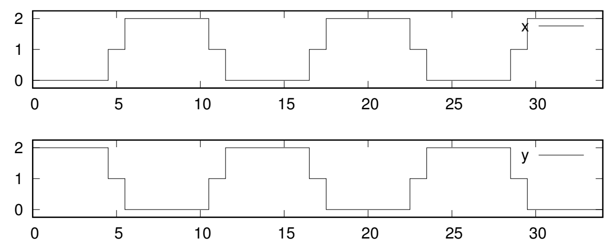

Figure 7: Example of signal.

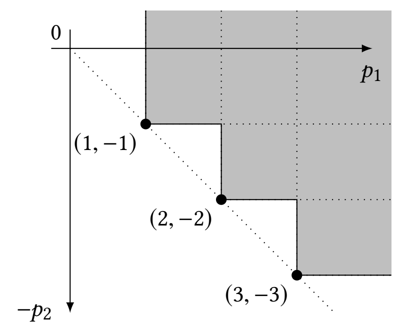

Figure 8: Validity domain at time 0 for the signal in Fig. 7 and the formula $\Diamond(x \leq p_1 \wedge x \geq p_2)$.

Range of Values. Perhaps the simplest use of PSTL is to find the bounds on the value of a signal x. For that, we can use the formula $\Box(x \leq p_1 \wedge x \geq p_2)$. Our procedure finds the smallest possible value of p_1 and the largest possible value of p_2 that renders the formula true, which gives for this signal the validity domain $(p_1 \geq 2 \wedge p_2 \leq 0)$ at time 0, meaning that the value of x lies between 0 and 2.

Enumeration of Values. Consider now the formula $\Diamond(x \leq p_1 \wedge x \geq p_2)$. For the example signal, the validity domain of this formula at time 0 is $(p_1 \geq 0 \wedge p_2 \leq 0) \vee (p_1 \geq 1 \wedge p_2 \leq 1) \vee (p_1 \geq 2 \wedge p_2 \leq 2)$. We show it in Fig. 8, following the convention of negating the values of the negative parameter p_2.

This validity domain actually enumerates all possible values of x. If we apply this formula to a sampled analog signal we may have a fast-growing set of incomparable rectangles. This is expensive to compute yet the outcome is not very informative.

Common Threshold. We can use the formula $\Box(x \geq p \vee y \geq p)$ to find the common threshold, s.t. at all times at least one signal component is above it. For our example the validity domain at time 0 is $(p \leq 1)$ meaning that at all times x is above 1 or y is above 1.

Hi/lo values. As a less simple example, we consider an analog signal x in whose value is interpreted as Boolean 1 if x is above some threshold x_{hi}, as Boolean 0 of x is below some threshold x_{lo}, and is considered transient otherwise, which can happen during a rising or a falling edge. Our example signal is a simple instance of this case, where $x_{\text{hi}} = 2$ and $x_{\text{lo}} = 0$. We can use PSTL to find the values of x_{hi} and x_{lo}, e.g., as follows. Let us assume we know the maximum duration of a rising or a falling edge t_{edge} and the minimum amount of time t_{stab} that the signal will spend in a well defined Boolean state after an edge. Then, we can use the formula $\Box\,\Diamond_{[0,\,t_{\text{edge}}+t_{\text{stab}}]}\left((\Box_{[0,\,t_{\text{stab}}]}\,x \le p_1) \vee (\Box_{[0,\,t_{\text{stab}}]}\,x \ge p_2)\right)$. For our example signal in Fig. 7, the validity domain of this formula at time 0 is the set $(p_1 \ge 2) \vee (p_1 \ge 0 \wedge p_2 \le 2) \vee (p_2 \le 0)$. In particular, we are interested in the rectangle that has the form $(p_2 \le x_{\text{hi}} \wedge p_1 \ge x_{\text{lo}})$ where $x_{\text{hi}} > x_{\text{lo}}$, i.e., the rectangle $(p_1 \ge 0 \wedge p_2 \le 2)$, which allows to conclude that $x_{\text{hi}} = 2$ and $x_{\text{lo}} = 0$, as expected. The other two rectangles in the validity domain are not relevant for our question. In practice, depending on the signal, we may have to apply the outer *always* with the time window $[0, |w| - t_{\text{edge}} - t_{\text{stab}}]$. If the signal ends in a transient state, the final time points will fail to satisfy the *eventually* subformula, and we will want to exclude them from the computation.

5 PERFORMANCE

In this section we give a preliminary evaluation of the performance of our algorithm from both theoretical and empirical perspectives, the latter based on our implementation of the identification procedure in OCaml.

Our backshifting algorithm can actually be seen as a modification of Lemire's algorithm [32] for computing minima and maxima over a shifting window, which, e.g., is used for robustness computation in [18]. The novel feature of our algorithm is that we work in a partially-ordered parameter space and we do union and intersection of validity domains (represented by Pareto sets of minimal supporting points) instead of min and max.

During backshifting, the tail of the intermediate result *res* is arranged in descending order (when f is set union, and in ascending order when f is set intersection) and plays the role of Lemire's queue. Thus, when the values of the validity signal (i.e., validity domains) that is being backshifted are totally ordered, backshifting performs $O(n)$ set operations (union, intersection), where n is the length of the signal. This is, e.g., the case when the validity domains have one dimension. *Until* and backshifting with a large upper bound (when *BackshiftInit* does all the work) take $O(n)$ set operations regardless of the structure of validity domains.

When some validity domains are incomparable, backshifting performs $O(mn)$ set operations where m is the maximum number of incomparable elements that fall within the same backshifting window. The value of m depends on the width of the backshifting window, but also comes from some property of the input signal. In favorable cases, it may be the period of a periodic signal or the width of raising and falling edges; in the worst case this may be the number of distinct values that the signal takes (see Section 4 for an example).

With our current implementation, the worst-case complexity of operations on sets of support points is $O(l^2)$ where l is the number of points in a set. For dimensions 1 and 2, this can be improved to $O(l)$ by storing the points in a sorted array, and there also exist tree structures suitable for higher dimensions. We do not have a good intuition into how l is connected to the length of the signal, n. It is easy to construct examples, where l is proportional to the length of the signal, but it is unclear whether it can grow faster. Also, in this work we focus on formulas and signals, for which l is small and does not depend too much on temporal windows in the formula, the length of the signal, presence of noise in it, etc.

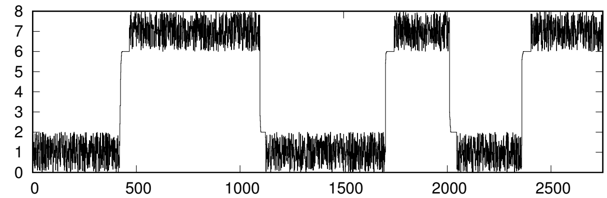

Figure 9: Fragment of a generated square wave with noise that we use in the evaluation.

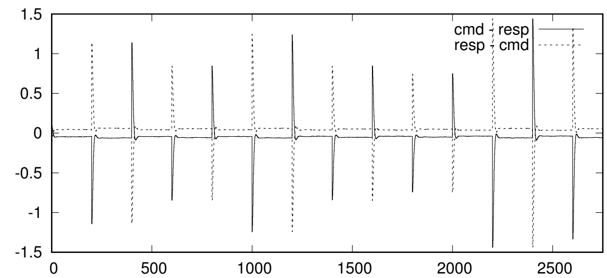

Figure 10: Fragment of the output of an airplane pitch control model. We show the difference between the command and response signals.

Artificially Generated Signals. In the first group of experiments, we evaluate the implementation using artificially generated signals of different length, and we have two kinds of signals. Signals of the first kind, referred as w_{sincos}, have two components: a sine wave $w_x[t] = \sin(\frac{2\pi t}{500})$ and a cosine wave $w_y[t] = \cos(\frac{2\pi t}{500})$. Both are sampled with a step of one time unit and thus have a period of 500 samples. Signals of the second kind, referred as w_{square}, are randomly generated square waves with noise and non-zero duration of the rising and falling edges. The average period is 1000 units, and again we take one sample per time unit. A fragment of this signal is shown in Fig. 9. We generate three versions of each signal with varying length: 10 000, 100 000, and 1 million samples.

Using these signals, we run parameter identification for a number of formulas and report the results (runtime of the identification procedure in seconds and, space permitting, the validity domain at time 0 for the signal with 1 million samples) in Table 1. The performance figures were obtained in a virtual machine running on a Core i7-3630QM laptop computer.

In this evaluation we are interested in signals and formulas for which the validity domains have a small number of rectangles that does not depend too much on the temporal windows in the formula, the length of the signal, presence of noise and so on; in particular, we wish to avoid validity domains that enumerate distinct signal values. The only formula in our evaluation which is ill-behaved in this sense is φ_6; its validity domain may depend on the presence of noise and also on the length of the signal.

The goal of the implementation was to efficiently perform parameter identification for formulas with small validity domains, and we think that we achieved this goal. For most example formulas, it takes a few seconds to run parameter identification for a signal

Table 1: Evaluation results.

Formula	Signal	$d(\varphi, w)[0]$	Time (in seconds)		
			10^4	10^5	10^6
$\varphi_1 = \Box(x \le p_1 \land x \ge p2)$	w_{sincos}	$p_1 \ge 1 \land p_2 \le -1$	0.03	0.36	3.9
	w_{square}	$p_1 \ge 8 \land p_2 \le -2.6 \cdot 10^{-6}$	0.03	0.36	3.75
$\varphi_2 = \Box(x \ge p \land y \ge p)$	w_{sincos}	$p \le -0.70265$	0.025	0.28	3.4
$\varphi_3 = \Box(y \ge p\, \mathcal{U}\, x \ge p)$	w_{sincos}	$p \le -0.70265$	0.03	0.31	3.4
$\varphi_4 = \Box(x \le 6 \Rightarrow \Diamond_{[0,50]}(x \ge 6 \lor x \le p))$	w_{square}	$p \ge 1.65696$	0.01	0.12	1.4
$\varphi_{5,1} = \Diamond_{[0,5K]}(x \ge p_1 \lor \Box_{[0,250]}\, y \ge p_2)$	w_{sincos}	$p_1 \le 1 \lor p_2 \le -2.4 \cdot 10^{-16}$	0.04	0.41	4.4
$\varphi_{5,2} = \Diamond_{[0,50K]}(x \ge p_1 \lor \Box_{[0,250]}\, y \ge p_2)$		$p_1 \le 1 \lor p_2 \le 4 \cdot 10^{-15}$	0.03	0.4	4.3
$\varphi_{6,1} = \Box_{[0,5K]}\, \Diamond_{[0,250]}((\Box_{[0,200]}\, x \le p_1) \lor (\Box_{[0,200]}\, x \ge p_2))$	w_{sincos}	not shown	0.24	4	44
	w_{square}	not shown	0.05	0.75	8
$\varphi_{6,2} = \Box_{[0,50K]}\, \Diamond_{[0,250]}((\Box_{[0,200]}\, x \le p_1) \lor (\Box_{[0,200]}\, x \ge p_2))$	w_{sincos}	not shown	0.08	2.5	42
	w_{square}	not shown	0.04	0.67	8.7
$\varphi_{6,3} = \Box_{[0,50K]}\, \Diamond_{[0,125]}((\Box_{[0,200]}\, x \le p_1) \lor (\Box_{[0,200]}\, x \ge p_2))$	w_{sincos}	not shown	2.6	TO	TO
	w_{square}	not shown	0.05	0.66	8.1
$\varphi_7 = \Box\, \Diamond_{[0,45]}\, \Box_{[0,30]}(x_{cmd-resp} \le p_1 \land x_{resp-cmd} \le p_2)$	w_{airplane}	$p_1 \le -0.066 \land p_2 \ge 0.066$	0.05	0.56	6

with 1 million samples, and the runtime grows linearly with the length of the signal or close to that.

We evaluate multiple versions of the formulas φ_5 and φ_6, with different temporal windows. Formulas $\varphi_{5,1}, \varphi_{5,2}$ and $\varphi_{6,1}, \varphi_{6,2}$ demonstrate the effect of changing the window of an outermost temporal operator in a formula with a small validity domain. We can observe that for shorter signals it is more efficient to backshift with a higher upper bound, since in the implementation *BackshiftInit* is more efficient than repeated application of *BackshiftInit*. Formula $\varphi_{6,3}$ demonstrates how changing the window of a nested temporal operator can change the formula from being well-behaved to ill-behaved for a given signal. There appears to be an interplay between the upper bounds of eventually and the signal period (250 corresponds to the half-period, 125 is the quarter-period).

Airplane Pitch Control Model. We also evaluate the implementation using the data produced by the Simulink model of an airplane longitudinal flight control system. The aim of this system is to control the *pitch* orientation of the airplane. We assume the control is biased with some fixed offset, and the expected property of the system is that within a delay of 15 units the response signal of the system stabilizes close to the command signal, and the stable state is kept for at least 30 units. The model is driven with a command signal that is piecewise-constant with constant periods lasting of 200 units (long enough for the response to stabilize), and varying command values in the range $[-1.0, 1.0]$. In Fig. 10 we show a fragment of the difference signal between command and response. In Table 1, we refer to this signal as w_{airplane} In the evaluation, we measure the bounds on the difference between command and response in a stable state. We use the fact response always stabilizes above command and employ the following formula: $\Box\, \Diamond_{[0,45]}\, \Box_{[0,30]}(x_{cmd-resp} \ge p_1 \land x_{resp-cmd} \le p_2)$. The validity domain of this formula at time 0 is $(p_1 \le -0.066 \land p_2 \ge 0.066)$, which means that response stabilizes within the range of 0.066 from command signal.

6 RELATED WORK

The problem of parameter estimation/synthesis in system models [15] as well as in properties such as those expressed in PSTL, is a crucial problem in the design and analysis of systems and is implemented in tools such as S-Taliro [5] and Breach [16]. The industrial-size case studies of [27] and [36] demonstrate the practical relevance of PSTL, and its value in conjunction with other methods.

In general, the problem that we solve in this paper can be seen as a case of learning from positive examples since we observe only behaviors which are possible. In other contexts, such as those of [13] and [11, 30], the traces can be classified as normal or abnormal and the problem becomes that of learning from both positive and negative examples, a case of *supervised* learning.

The works of [26], of [25, 42], and of [29] are most related to ours. The problem they study can be stated as follows: given a system model M and a parameterized temporal formula $\varphi[p]$, find the validity domain of φ relative to M, that is, the set of parameter valuations v such that $w \models \varphi[v]$. Here the system model is viewed as a black box, that can produce from some input $u \in U$ a simulation trace $w = M(u)$. Since the input space U is typically large or uncountable, such methods are inherently approximate. They explore the boundary of

$$D(\varphi, M) = \bigcap_{w \in M(U)} D(\varphi, w)$$

by search in the combined space $U \times V$ of system inputs and formula parameters.

Our work builds up on that of [6] to solve the problem, for a single simulation or execution trace of the system. In fact [6] considered two techniques, one exact and based on translation to a quantified formula in linear arithmetic, and one approximate, based on search in the parameter space. In this work, we solve the problem in an exact manner using signal-processing computations in the style of [18, 33], avoiding reliance on a (costly) quantifier elimination routine. The exhaustive computation of the validity domain is similar to that of [12], who study the problem of monitoring STL*, a variant of STL with *freeze* quantification [4]. There the parameter space is over subsets of $\mathbb{T}^k$ for k frozen variables, but could be rephrased over $\mathbb{R}^k$, something we intend to explore further.

With or without parameters, the main application of STL monitoring is found in *falsification*. This problem, dual of verification, attempts not to prove that the system M is correct under all inputs $u \in U$, but simply to find a faulty execution $w = M(u)$, without any formal guarantees that it will be found. The most effective technique, as illustrated in [2, 34], turns the falsification problem into the following optimization problem:

$$\min \rho(\varphi, w) \qquad \text{s.t.} \qquad u \in U, w = M(u)$$

The robustness value ρ is expected to be continuous in u, and by definition is $w \not\models \varphi$ when $\rho(\varphi, w) < 0$. Indeed many works [25, 27, 29, 42] use this technique in the setting of temporal logic parameter exploration.

The choice of minimization algorithm in the above is crucial, and several alternatives have been explored, see [1, 39] for instance. It can be argued with [3, 26] that the choice of the cost function is equally important. An undesirable behavior of the robustness value as cost function, is due to its absolute-norm semantics, which selects the value of the safest signal variable (the furthest away from violation) as the one to optimize. We believe that parameterizations of STL formulas provide a way to define better behaved (smoother) cost functions. For this, observe that the robustness $\rho(\varphi, w)$ can be recovered as the tightest parameter assignments of PSTL formula $\varphi'[p]$ where p is a unique parameter replacing all constants, and φ' is the positive normal form of φ. Using several parameters (for several signal variables) could provide a cost function with non-zero derivative in more than one variable.

7 CONCLUSIONS AND FUTURE WORK

In this work we presented a novel algorithm for parametric identification for STL and applied it to formulas with space parameters and piecewise-constant (PC) signals where the validity domains are union of rectangles. We have shown that in many cases, a prototype implementation of our algorithm can compute the validity domain for signals with hundreds of thousands to millions of samples. Hence our methods provides a viable alternative to previously-used algorithms based on quantifier elimination or search in the parameter-space, and will allow the derivation of compact representations of systems based on observable behaviors.

The immediate direction for future work is improving the implementation of the algorithm and in particular the representation of validity domains. Currently, we store validity domains as unsorted arrays of support points, thus the complexity of disjunction and conjunction is quadratic in the size of the support set. More efficient representations of such sets are known in the context of maintaining a Pareto front, but adapting the data structures and algorithms to our setting is still a challenge.

There are two immediate extensions of our work. If we use a piecewise-linear (PL) interpolation for signals, we will not have rectangular validity domains in each intervals but polytopes that depend on t. The computational trade-offs between using PL and PC signals should be investigated. On one hand, rectangles are easier to manipulate but on the other, PC signals will require a denser sampling than PL signals to achieve the same approximation level with respect to the underlying continuous signal. In general the influence of sampling rates of the same signal on the obtained validity domain should be studied.

The more interesting and urgent extension is to include timing parameters where, naturally, validity domains will explicitly depend on time, which changes continuously and does not immediately fit in the piecewise-constant setting. We are confident, though, that this can be done and that the validity domains associated with timing parameters and PC signals are more restricted types of polyhedral sets than the semi-linear sets associated with PL signals. It should be noted that a mixture of space and time parameters may lead to many incomparable points. For example, the validity domain for the PSTL formula $\Diamond_{[0,p_1]} x \leq p_2$ relative to a decaying signal consists of a continuum of incomparable (p_1, p_2) points. Another question is whether our approach can be translated to other timed formalisms such as signal regular expressions [7].

Parametric validity domains can be viewed as a multi-dimensional generalization of robustness which we suggest will provide finer information concerning the robustness associated with subformulas and with different parts of the signal. Some work is needed to gain better insights on the precise relation between the two and its implications for robustness guided falsification algorithms.

Finally, in addition to providing a succinct representation of observed behaviors, we can view PSTL formulas as a new type of *feature extractors*, functions that map high-dimensional objects such as signals into low-dimensional objects, in our case sets of tightest values in the validity domain. Once mapped into this space, the signals can be subject to various learning and clustering algorithms as suggested recently in [41].

REFERENCES

[1] Houssam Abbas and Georgios E. Fainekos. 2013. Computing descent direction of MTL robustness for non-linear systems. In *American Control Conference*. 4405–4410. http://ieeexplore.ieee.org/document/6580518/

[2] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, and Koichi Ueda. 2014. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*. IEEE, 1–6.

[3] Takumi Akazaki and Ichiro Hasuo. 2015. Time robustness in MTL and expressivity in hybrid system falsification. In *CAV*. Springer, 356–374.

[4] Rajeev Alur and Thomas A Henzinger. 1994. A really temporal logic. *Journal of the ACM (JACM)* 41, 1 (1994), 181–203.

[5] Yashwanth Annpureddy, Che Liu, Georgios E Fainekos, and Sriram Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems.. In *TACAS*, Vol. 6605. Springer, 254–257.

[6] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. 2011. Parametric identification of temporal properties. In *Runtime Verification*. Springer, 147–160.

[7] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. 2017. On the Quantitative Semantics of Regular Expressions over Real-Valued Signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 189–206.

[8] Ezio Bartocci, Luca Bortolussi, and Laura Nenzi. 2013. A temporal logic approach to modular design of synthetic biological circuits. In *International Conference on Computational Methods in Systems Biology*. Springer, 164–177.

[9] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. 2018. Specification-based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In *The Handbook of Runtime Verification*.

[10] Giuseppe Bombara and Calin Belta. 2017. *Signal Clustering Using Temporal Logics*. Springer International Publishing, Cham, 121–137. DOI:http://dx.doi.org/10.1007/978-3-319-67531-2_8

[11] Giuseppe Bombara, Cristian-Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. 2016. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *HSCC*. ACM, 1–10.

[12] Lubos Brim, P Dluhoš, D Šafránek, and Tomas Vejpustek. 2014. STL*: Extending signal temporal logic with signal-value freezing operator. *Information and Computation* 236 (2014), 52–67.

[13] Sara Bufo, Ezio Bartocci, Guido Sanguinetti, Massimo Borelli, Umberto Lucangelo, and Luca Bortolussi. 2014. Temporal Logic Based Monitoring of Assisted Ventilation in Intensive Care Patients. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II*. 391–403. DOI: http://dx.doi.org/10.1007/978-3-662-45231-8_30

[14] Fraser Cameron, Georgios E. Fainekos, David M. Maahs, and Sriram Sankaranarayanan. 2015. Towards a Verified Artificial Pancreas: Challenges and Solutions for Runtime Verification. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*. 3–17. DOI: http://dx.doi.org/10.1007/978-3-319-23820-3_1

[15] Thao Dang, Tommaso Dreossi, and Carla Piazza. 2015. Parameter synthesis through temporal logic specifications. In *International Symposium on Formal Methods*. Springer, 213–230.

[16] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems.. In *CAV*, Vol. 10. Springer, 167–170.

[17] Alexandre Donzé, Eric Fanchon, Lucie Martine Gattepaille, Oded Maler, and Philippe Tracqui. 2011. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PloS one* 6, 9 (2011), e24246.

[18] Alexandre Donzé, Thomas Ferrere, and Oded Maler. 2013. Efficient robust monitoring for STL. In *CAV*. 264–279.

[19] A. Donzé, B. Krogh, and A. Rajhans. 2009. Parameter synthesis for hybrid systems with an application to simulink models. In *HSCC (LNCS)*. Springer-Verlag.

[20] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *FORMATS*. 92–106.

[21] Georgios E. Fainekos and Georges J. Pappas. 2006. Robustness of Temporal Logic Specifications. In *FATES/RV (LNCS)*, Vol. 4262. Springer, 178–192.

[22] 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.

[23] Samira S Farahani, Vasumathi Raman, and Richard M Murray. 2015. Robust model predictive control for signal temporal logic synthesis. *IFAC-PapersOnLine* 48, 27 (2015), 323–328.

[24] Carlo A Furia and Matteo Rossi. On the expressiveness of MTL variants over dense time. Springer.

[25] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. 2017. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer* (2017), 1–15.

[26] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. 2017. TeLEx: Passive STL Learning Using Only Positive Examples. In *Runtime Verification*. 208–224.

[27] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. 2013. Mining Requirements from Closed-loop Control Models. In *HSCC*.

[28] Kevin D Jones, Victor Konrad, and Dejan Nickovic. 2010. Analog property checkers: a DDR2 case study. *Formal Methods in System Design* 36, 2 (2010), 114–130.

[29] Eric S Kim, Murat Arcak, and Sanjit A Seshia. 2016. Directed specifications and assumption mining for monotone dynamical systems. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 21–30.

[30] Zhaodan Kong, Austin Jones, and Calin Belta. 2017. Temporal logics for learning and detection of anomalous behavior. *IEEE Trans. Automat. Control* 62, 3 (2017), 1210–1222.

[31] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. 2010. Approximating the Pareto Front of Multi-criteria Optimization Problems. In *TACAS (LNCS)*, Vol. 6015. Springer, 69–83.

[32] D. Lemire. 2006. Streaming Maximum-Minimum Filter Using No More than Three Comparisons per Element. *CoRR* abs/cs/0610046 (2006).

[33] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *FORMATS/FTRTFT*. 152–166.

[34] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J Pappas. 2010. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *HSCC*. ACM, 211–220.

[35] Dejan Nickovic. 2008. *Checking timed and hybrid properties: Theory and applications*. Ph.D. Dissertation. Université Joseph Fourier, Grenoble, France.

[36] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B Finn, Alberto L Sangiovanni-Vincentelli, Richard M Murray, Alexandre Donzé, and Sanjit A Seshia. 2014. A contract-based methodology for aircraft electric power system design. *IEEE Access* 2 (2014), 1–25.

[37] 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*. ACM, 239–248.

[38] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. 2008. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *CMSB*. Springer, 251–268.

[39] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. 2017. An Active Learning Approach to the Falsification of Black Box Cyber-Physical Systems. *arXiv preprint arXiv:1705.01879* (2017).

[40] Szymon Stoma, Alexandre Donzé, François Bertaux, Oded Maler, and Gregory Batt. 2013. STL-based analysis of TRAIL-induced apoptosis challenges the notion of type I/type II cell line classification. *PLoS computational biology* 9, 5 (2013), e1003056.

[41] Marcell Vazquez-Chanlatte, Jyotirmoy V. Deshmukh, Xiaoqing Jin, and Sanjit A. Seshia. 2017. *Logical Clustering and Learning for Time-Series Data*. Springer International Publishing, Cham, 305–325. DOI: http://dx.doi.org/10.1007/978-3-319-63387-9_15

[42] Hengyi Yang, Bardh Hoxha, and Georgios E Fainekos. 2012. Querying Parametric Temporal Logic Properties on Embedded Systems. In *ICTSS*. Springer, 136–151.