

Learning Auditable Features from Signals Using Unsupervised Temporal Projection

Marcell V-Chanlatte
University of California
Berkeley
marcell.vc@berkeley.edu

Xiaoqing Jin
Toyota Technical Center
xiaoqing.jin@toyota.com

Jyotirmoy V. Deshmukh
Toyota Technical Center
jyotirmoy.deshmukh@toyota.com

Sanjit Seshia
University of California
Berkeley
sseshia@eecs.berkeley.edu

ABSTRACT

To effectively analyze and design cyberphysical systems (CPS), designers today have to combat the data deluge problem, *i.e.*, the burden of processing intractably large amounts of data produced by complex models and experiments. In this work, we utilize monotonic Parametric Signal Temporal Logic (PSTL) to design features for unsupervised classification of time series data. This enables using off-the-shelf machine learning tools to automatically cluster similar traces with respect to a given PSTL formula. We demonstrate how this technique produces simple and interpretable formulas that are amenable to analysis and understanding using a few representative examples. We illustrate this with a number of case studies related to automotive engine testing, highway traffic analysis, and auto-grading massively open online courses.

1. INTRODUCTION

To effectively analyze and design cyberphysical systems (CPS), designers today have to combat the *data deluge* problem, *i.e.*, the burden of processing intractably large amounts of data produced by complex models and experiments. For example, consider the typical design process for a cutting edge CPS such as a self-driving car. Checking whether the car meets all its requirements is typically done by either physically driving the car around for millions of miles [2], or by virtual simulations of the self-driving algorithms. Either scenario generates several gigabytes worth of time-series data with the potential to reveal suboptimal performance of the underlying software algorithms or driving scenarios not adequately addressed by the software. However, patterns encoding such interesting behaviors are often not known *a priori*, and identifying such patterns may require manual, visual inspection of the timed traces. As another example, consider the problem of monitoring the operation of a CPS in an online fashion; for example, a pacemaker monitoring car-

diac signals [5, 1]. In such a setting, the underlying software may have to be trained to automatically recognize anomalous behavior in the ECG signals and perform corrective action.

One approach for automatic recognition of patterns and trends in timed traces is to employ similarity metrics to cluster signal traces. The most popular such metric is the dynamic time warping (DTW) distance [16]. While the DTW distance can generate clusters with qualitatively similar signal shapes, there are a few shortcomings. First, to interpret the logical properties that signal traces in a cluster may satisfy requires doing additional analysis on a cluster [14]. Second, it may be difficult for the metric to distinguish between signals that are qualitatively similar in shape but differ only in their temporal behavior.

An alternative is to consider techniques that project time-series signals onto a user-defined feature space, and then perform the learning on the feature space. This is an attractive option, as it reduces learning on an infinite-dimensional space of signals to a finite-dimensional space of features. Several different feature-extraction based methods have been studied in the literature such as those based on frequency-domain transformations, statistical measures, and those based on autoregressive models [20]. In specific cases, such feature spaces can help the designer interpret the results of the learning algorithms at a logical level, and thus gain insights.

In this paper, we explore unsupervised learning for timed traces using a feature extraction procedure based on *Parametric Signal Temporal Logic* (PSTL) templates¹. . Our motivation for choosing PSTL for feature extraction comes from techniques used to help CPS designers perform early validation of their designs. Here, the use of real-time temporal logics such as Signal Temporal Logic (STL) to specify correctness and performance requirements of designs has been gathering steam. A big hurdle is that control designers are rarely trained to express requirements using logical formalisms. To improve this situation, in [11], the authors identified a set of Parametric STL (PSTL) templates that capture a number of critical time-varying behaviors from a control-theoretic perspective. Specifying a range of values for the parameters in a PSTL template effectively specifies a set of signals satisfying the template. Typically, parameters appearing in the template have physical meaning associated with them; for example, for a template specifying a “spike,”

HSCC 217 Pittsburgh, Pennsylvania USA

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

¹Note that the work presented in this paper naturally generalizes to any monotonic parametric logic.

parameter ranges may specify the minimum height or the maximum width that qualifies a signal to be considered as having a spike. This offers a natural advantage to the PSTL templates: they serve as a logical expression of the visual language with which the control designers are very familiar.

Projecting a signal through the lens of a PSTL template generates a feature space that is low-dimensional, yet has physical meaning. Once such projection is performed, traditional machine learning algorithms such as unsupervised clustering and classification for supervised learning can readily be employed. The result of such learning algorithms is defined in terms of *concepts* (which for our purpose are simply geometric subsets in the parameter space). In our approach a concept encloses a cluster as learned by a traditional clustering algorithm. The set of shapes allowed for concepts is called the concept class. Using PSTL-based feature extraction has the advantage that concepts learned in the feature space correspond to STL formulas. In this paper we show that for if the PSTL template has the *monotonicity* property, then we can restrict the concept class to concepts that can be expressed as a union of a bounded number of axis-aligned hyperboxes. This yields another advantage: such a concept class gives rise to STL formulas of only a bounded length and complexity. This point is particularly pertinent given anecdotal evidence that designer comprehension of a temporal logic formula automatically learned from traces is generally difficult. We argue that our choice of PSTL templates and our chosen concept class can allow designers to gain a better understanding of the STL formula learned. We base our argument on the fact that each concept learned can be explained using a small number of representative timed traces, and that any other timed trace belonging to the concept is related to the representative traces through simple operations.

1.1 Related Work

Seminal work on Temporal Logic Specification Mining has been proposed for STL in a supervised learning context[6, 17, 4], in the unsupervised anomaly detection context[13], and in the context of active learning[14]. One approach has been to adapt classical machine learning algorithms such as decision trees[6] and One-Class Support Vector Machines[13]. The advantage of these approaches is that they do not restrict the length of the STL formulas considered. However, in some cases, this can be a disadvantage as long formulas may not admit an easy interpretation by the designer. More pragmatically, these techniques require the explicit adaptation of classic machine algorithms.

To address the labeling problem one may consider first clustering in the trace space (using Dynamic Time Warping (DTW) for example) along with active learning to characterize the validity domain of a given PSTL formula[14]. We distinguish our work by first noting that, by itself, DTW is not guaranteed to group signals under the lens of the given PSTL template. For example consider a simple template $\mathbf{G}(x < c)$. Under this coarse lens, any signals with similar maximum values should be grouped together. Dynamic Time warping will only group together signals that are nearly time/spatial distortions of each other. Second, our STL extraction technique does not presuppose a binary classification. While sufficient in many contexts, one of our goals is to characterize sub-populations in the dataset. In section 6.3 we reproduce a subset of the results in [14] to qualitatively compare and contrast the techniques.

2. PRELIMINARIES

DEFINITION 2.1 (TIMED TRACES). *A timed trace is a finite (or infinite) sequence of pairs $(t_0, \mathbf{x}_0), \dots, (t_n, \mathbf{x}_n)$, where, $t_0 = 0$, for all $i \in [1, n]$, $t_i \in \mathbb{R}_{\geq 0}$, $t_{i-1} < t_i$, and for all $i \in [0, n]$, $\mathbf{x}_i \in \mathcal{D}$, where $\mathcal{D}$ is some compact set. We refer to the interval $[t_0, t_n]$ as the time domain T .*

Real-time temporal logics are an effective formalism to reason about finite or infinite timed traces. Logics such as the Timed Propositional Temporal Logic [3], and Metric Temporal Logic (MTL) [18] were introduced to reason about signals representing Boolean-predicates varying over dense (or discrete) time. More recently, Signal Temporal Logic [21] was proposed in the context of analog and mixed-signal circuits as a specification language for constraints on real-valued signals.

Signal Temporal Logic. Without loss of generality, atoms in STL formulas can be reduced to the form $f(\mathbf{x}) \sim c$, where f is a function from $\mathcal{D}$ to $\mathbb{R}$, $\sim \in \{\geq, \leq, =\}$, and $c \in \mathbb{R}$. Temporal formulas are formed using temporal operators, “always” (denoted as $\mathbf{G}$), “eventually” (denoted as $\mathbf{F}$) and “until” (denoted as $\mathbf{U}$) that can each be indexed by an interval I . An STL formula is written using the following grammar:

$$\begin{aligned} I &:= (a, b) \mid (a, b] \mid [a, b) \mid [a, b] \mid [a, \infty) \mid (a, \infty) \\ \varphi &:= \text{true} \mid f(\mathbf{x}) \sim c \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \end{aligned} \quad (2.1)$$

In the above grammar, $a, b \in T$, and $c \in \mathbb{R}$. The always ($\mathbf{G}$) and eventually ($\mathbf{F}$) operators are defined for notational convenience, and are just special cases of the until operator: $\mathbf{F}_I \varphi \triangleq \text{true} \mathbf{U}_I \varphi$, and $\mathbf{G}_I \varphi \triangleq \neg \mathbf{F}_I \neg \varphi$. We use the notation $(\mathbf{x}, t) \models \varphi$ to mean that the suffix of the timed trace $\mathbf{x}$ beginning at time t satisfies the formula φ . Formally, the semantics of an STL formula are defined recursively:

$$\begin{aligned} (\mathbf{x}, t) \models f(\mathbf{x}) \sim c &\iff f(\mathbf{x}(t)) \sim c \text{ is true} \\ (\mathbf{x}, t) \models \neg \varphi &\iff (\mathbf{x}, t) \not\models \varphi \\ (\mathbf{x}, t) \models \varphi_1 \wedge \varphi_2 &\iff (\mathbf{x}, t) \models \varphi_1 \wedge (\mathbf{x}, t) \models \varphi_2 \\ (\mathbf{x}, t) \models \varphi_1 \mathbf{U}_I \varphi_2 &\iff \exists t_1 \in t \oplus I : (\mathbf{x}, t_1) \models \varphi_2 \wedge \\ &\quad \forall t_2 \in (t, t_1) : (\mathbf{x}, t_2) \models \varphi_1 \end{aligned}$$

We write $\mathbf{x} \models \varphi$ as a shorthand of $(\mathbf{x}, 0) \models \varphi$.

Parametric Signal Temporal Logic (PSTL). PSTL [4] is an extension of STL introduced to define *template formulas* containing unknown parameters. Formally, the set of parameters $\mathcal{P}$ is a set consisting of two disjoint sets of variables $\mathcal{P}^V$ and $\mathcal{P}^T$ of which at least one is nonempty. The parameter variables in $\mathcal{P}^V$ can take values from their domain, denoted as the set V . The parameter variables in $\mathcal{P}^T$ are time-parameters that take values from the time domain T . We define a valuation function ν that maps a parameter to a value in its domain. We denote a vector of parameter variables by $\mathbf{p}$, and extend the definition of the valuation function to map parameter vectors $\mathbf{p}$ into tuples of respective values over V or T . We define the *parameter space* $\mathcal{D}_{\mathcal{P}}$ as a subset of $V^{|\mathcal{P}^V|} \times T^{|\mathcal{P}^T|}$.

A PSTL formula is then defined by modifying the grammar specified in (2.1) by allowing a, b to be elements of $\mathcal{P}^T$, and c to be an element of $\mathcal{P}^V$. An STL formula is obtained by pairing a PSTL formula with a valuation function that assigns a value to each parameter variable. For example, consider the PSTL formula $\varphi(c, \tau) = \mathbf{G}_{[0, \tau]} x > c$, with parameters variables c and τ . The STL formula $\mathbf{G}_{[0, 10]} x > 1.2$ is an instance of φ obtained with the valuation $\nu = \{\tau \mapsto 10, c \mapsto 1.2\}$.

DEFINITION 2.2 (VALIDITY DOMAIN). *Let X be a finite or infinite collection of timed traces, and let $\varphi(\mathbf{p})$ be a*

PSTL formula with parameters $\mathbf{p} \in \mathcal{P}$. The validity domain $\mathcal{V}(\varphi(\mathbf{p}))$ of $\varphi(\mathbf{p})$ is a closed subset of $\mathcal{D}_{\mathcal{P}}$, such that:

$$\forall \nu(\mathbf{p}) \in \mathcal{V}(\varphi(\mathbf{p})) : \forall \mathbf{x} \in X : \mathbf{x} \models \varphi(\nu(\mathbf{p})) \quad (2.2)$$

The validity domain for a given parameter set $\mathcal{P}$ essentially contains all the parameter valuations such that for the given set of traces X , each trace satisfies the STL formula obtained by instantiating the given PSTL formula with the parameter valuation. The boundary of the validity domain in the topological sense (denoted $\partial\mathcal{V}(\varphi(\mathbf{p}))$) is defined in standard fashion – we require that a valuation $\nu(\mathbf{p})$ that lies in the set $\partial\mathcal{V}(\varphi(\mathbf{p}))$ has the property that its every neighborhood contains at least one valuation (say ν') such that for some trace $\mathbf{x} \in X$, $\mathbf{x} \not\models \varphi(\nu'(\mathbf{p}))$. We also call such a valuation a *tight* valuation.

Technical Note. : If $\mathcal{V}(\varphi(\mathbf{p})) = \mathcal{D}_{\mathcal{P}}$, then we define $\partial\mathcal{V}(\varphi(\mathbf{p})) \triangleq \{\top\}$. Similarly, if $\mathcal{V}(\varphi(\mathbf{p})) \cap \mathcal{D}_{\mathcal{P}} = \emptyset$ then $\partial\mathcal{V}(\varphi(\mathbf{p})) \triangleq \{\perp\}$. The interpretation is that we find the point in $\mathcal{D}_{\mathcal{P}}$ “closest” to disatisfaction or satisfaction resp.

Monotonic PSTL. Monotonic PSTL is a fragment of PSTL introduced as the polarity fragment in [4]. A PSTL formula φ is said to be monotonically increasing in parameter p_i if condition (2.3) holds for all $\mathbf{x}$, and is said to be monotonically decreasing in parameter p_i if condition (2.4) holds for all $\mathbf{x}$.

$$(\mathbf{x} \models \varphi(\nu(p_i))) \wedge (\nu(p_i) \leq \nu'(p_i)) \implies \mathbf{x} \models \varphi(\nu'(p_i)) \quad (2.3)$$

$$(\mathbf{x} \models \varphi(\nu(p_i))) \wedge (\nu(p_i) \geq \nu'(p_i)) \implies \mathbf{x} \models \varphi(\nu'(p_i)) \quad (2.4)$$

To indicate the direction of monotonicity, we now introduce the polarity of a parameter [4], $\text{sgn}(p_i)$, and say that $\text{sgn}(p_i) = +$ if the $\varphi(\mathbf{p})$ is monotonically increasing in p_i and $\text{sgn}(p_i) = -$ if it is monotonically decreasing, and $\text{sgn}(p_i) = \perp$ if it is neither. A formula $\varphi(\mathbf{p})$ is said to be monotonic in p_i if $\text{sgn}(p_i) \in \{+, -\}$, and say that $\varphi(\mathbf{p})$ is monotonic if for all i , φ is monotonic in p_i .

While restrictive, the monotonic fragment of PSTL contains many formulas of interest, such as those expressing steps and spikes in trace values, timed-causal relations between traces, and so on. Moreover, in some instances, it may be possible to transform a non-monotonic PSTL formula to a monotonic PSTL formula by introducing new parameters or using a constant valuation for some parameters. We demonstrate with an example.

Example 1. Consider the PSTL template:

$$\varphi((h, w)) \stackrel{\text{def}}{=} \mathbf{F}((x \leq h) \wedge \mathbf{F}_{[0, w]}(x \geq h)) \quad (2.5)$$

It can be seen that the formula (2.5) is not monotonic. Observe however, that the following relaxed PSTL formula is trivially monotonic:

$$\varphi((w, h_1, h_2)) \stackrel{\text{def}}{=} \mathbf{F}((x \leq h_1) \wedge \mathbf{F}_{[0, w]}(x \geq h_2)) \quad (2.6)$$

Orders on Parameter Space. A monotonic parameter induces a total order $\leq_i$ in its domain, and as different parameters for a given formula are usually independent, valuations for different parameters induce a partial order. We formalize this below.

DEFINITION 2.3 (PARAMETER SPACE PARTIAL ORDER). We define $\leq_i$ as a total order on the domain of the parameter p_i as follows:

$$\nu(p_i) \leq_i \nu'(p_i) \stackrel{\text{def}}{=} \begin{cases} \text{if } \text{sgn}(p_i) = + & \text{then } \nu(p_i) \leq \nu'(p_i), \\ \text{if } \text{sgn}(p_i) = - & \text{then } \nu(p_i) \geq \nu'(p_i). \end{cases} \quad (2.7)$$

Under the order $\leq_i$, the parameter space can be viewed as a partially ordered set $(\mathcal{D}_{\mathcal{P}}, \leq)$, where the ordering operation $\leq$ is defined as follows:

$$\nu(\mathbf{p}) \leq \nu'(\mathbf{p}) \stackrel{\text{def}}{=} \forall i : \nu(p_i) \leq_i \nu'(p_i). \quad (2.8)$$

Validity domains for monotonic formulae have special structure: they are monotonic under the partial order $\leq$ [4].

DEFINITION 2.4 (UPWARD-CLOSED VALIDITY DOMAIN). The validity domain $\mathcal{V}(\varphi(\mathbf{p}))$ is said to be upward-closed if given $\nu(\mathbf{p}) \in \mathcal{V}(\varphi(\mathbf{p}))$, $\nu(\mathbf{p}) \leq \nu'(\mathbf{p})$ implies that $\nu'(\mathbf{p}) \in \mathcal{V}(\varphi(\mathbf{p}))$. For a valuation $\nu(\mathbf{p})$, we call the set $\{\nu'(\mathbf{p}) \mid \nu(\mathbf{p}) \leq \nu'(\mathbf{p})\}$ as the upward closure of the valuation $\nu(\mathbf{p})$.

Finally, we introduce Concept Classes from computational learning theory [15]. For our purposes, given a set A , a concept is any subset of A . A concept class, unsurprisingly, is a class/family of related concepts. Examples include Separating Hyperplanes, Axis Aligned Rectangles, Automata, etc. In this work, we will focus on geometric concept classes.

3. PSTL FOR TEMPORAL PROJECTION

In this section, we introduce the key idea of this paper: defining a projection operation π that maps a given timed trace $\mathbf{x}$ to a unique parameter valuation² $\nu^*(\mathbf{p})$ of a given Parametric Signal Temporal Logic formula $\varphi(\mathbf{p})$. We require that the projection function π to have two properties:

1. (Tightness) $\nu^*(\mathbf{p}) \in \partial\mathcal{V}(\varphi(\mathbf{p}))$, and,
2. (Uniqueness) $\forall \nu(\mathbf{p}) \in \partial\mathcal{V}(\varphi(\mathbf{p})), \nu^*(\mathbf{p}) \leq \nu(\mathbf{p})$.

Intuitively, we would like a projection operation to be as close a representative of the given timed trace as feasible (which is achieved by ensuring that the projection is tight), and we would like the projection operation to be canonical (which is achieved by ensuring uniqueness). In general, a tight projection can be obtained by picking any parameter valuation $\nu(\mathbf{p})$ in $\partial\mathcal{V}(\varphi(\mathbf{p}))$. One way of achieving a unique projection is by defining a total order $\preceq$ on the parameter space by an appropriate *linearization* of the partial order on parameter valuations. The total order then provides a unique minimum valuation that is then chosen as the result of projecting the timed trace onto the parameter space. We now explore a couple of such total orders.

Scalarization. Borrowing a common trick from multi-objective optimization, we define a cost function on the space of valuations as follows:

$$\text{cost}(\nu(\mathbf{p})) = \sum_{i=1}^{|\mathcal{P}|} \lambda_i \nu(p_i) \quad (3.1)$$

Here, $\lambda_i \in \mathbb{R}$, are weights on each parameter. The above cost function implicitly defines a total order $\preceq_{\text{scalar}}$, where, $\nu(\mathbf{p}) \preceq_{\text{scalar}} \nu'(\mathbf{p})$ iff $\text{cost}(\nu(\mathbf{p})) \leq \text{cost}(\nu'(\mathbf{p}))$. Then, the projection operation π_{scalar} is defined as follows³:

$$\pi_{\text{scalar}}(\mathbf{x}) = \underset{\nu(\mathbf{p}) \in \partial\mathcal{V}(\varphi(\mathbf{p}))}{\text{argmin}} \quad \text{cost}(\nu(\mathbf{p})) \quad (3.2)$$

²We note that in general, it is not necessary to define the projection to map a timed trace to a single point in the space of parameter valuations. For example, in some cases, it may be expedient to project a timed trace to the curve representing the validity domain boundary of a given PSTL formula. In other cases, we may want to project a timed trace to parameter spaces of multiple PSTL formulas. For simplicity, we defer more involved projections to a future exposition.

Lexicographic Order. A lexicographic order (denoted $\preceq_{\text{lex}}$) is another linearization of $\preceq$ that uses the specification of a total order on parameter indices to linearize the partial order. We formalize lexicographical ordering as follows.

DEFINITION 3.1 (LEXICOGRAPHIC ORDER). *Suppose we are given a total order on the parameters $j_1, \dots, j_n$, where each $p_{j_k} \in \mathcal{P}$. The total order $\preceq_{\text{lex}}$ on the parameter space $\mathcal{D}_{\mathcal{P}}$ is defined as:*

$$\begin{aligned} \nu(\mathbf{p}) \preceq_{\text{lex}} \nu'(\mathbf{p}) \text{ iff} \\ \exists j_k \in (j_1, \dots, j_n) \text{ s.t. } \nu(p_{j_k}) \preceq_i \nu'(p_{j_k}) \text{ and,} \\ \forall \ell < k, \nu(p_{j_\ell}) = \nu'(p_{j_\ell}). \end{aligned} \quad (3.3)$$

Note that for a given total or partial order, we can define inf and sup under that order in standard fashion. Formally, the projection function using lexicographic order is defined as follows:

$$\pi_{\text{lex}}(\mathbf{x}) = \inf_{\preceq_{\text{lex}}} \{ \nu(\mathbf{p}) \in \partial\mathcal{V}(\varphi(\mathbf{p})) \} \quad (3.4)$$

3.1 Computing Projections

In this section, we briefly review methods to compute projections. We first note that in 3.2 and 3.4, we require parameter valuations to lie exactly on the validity domain boundary. From a computational perspective, this is not practical. Hence, similar to [4], we define an ϵ -approximation of the validity domain boundary $\partial\mathcal{V}^\epsilon(\varphi(\mathbf{p}))$ as the largest such subset of $\mathcal{V}(\varphi(\mathbf{p}))$: $\{ \nu(\mathbf{p}) \mid \exists \nu'(\mathbf{p}) \in \partial\mathcal{V}(\varphi(\mathbf{p})) : \|\nu'(\mathbf{p}) - \nu(\mathbf{p})\| < \epsilon \}$

Computing π_{scalar} . For a multiobjective optimization problem where the cost function is a weighted sum of convex objectives and the feasible region is convex, efficient convex optimization algorithms can be used to obtain an optimal solution [22]. We observe from [4] that though the validity domain of a monotonic PSTL formula is upward-closed, this does not imply convexity of the validity domain. As a result, we are generally not guaranteed to find a projection using convex optimization methods. However, using an algorithm such as the one presented in [19], it is possible to obtain an approximation of $\partial\mathcal{V}^\epsilon(\varphi(\mathbf{p}))$, and solve Eq. (3.2) to obtain $\pi_{\text{scalar}}(\mathbf{x})$.

Computing π_{lex} . To compute $\pi_{\text{lex}}(\mathbf{x})$, we can use efficient algorithms to obtain a valuation in $\partial\mathcal{V}^\epsilon(\varphi(\mathbf{p}))$ for an arbitrarily small ϵ . We recall Algorithm 1 from [12] that uses a simple lexicographic binary search⁴.

We begin by first setting the interval to search for a valuation in $\partial\mathcal{V}^\epsilon(\varphi(\mathbf{p}))$. We set the initial valuation to $\perp$ since it induces the most permissive STL formula. Next, for each parameter, (in the order imposed by $\preceq_{\text{lex}}$), we perform bisection search on the interval to find a valuation in $\partial\mathcal{V}^\epsilon(\varphi(\mathbf{p}))$. Once completed, we return the lower bound of the search-interval as it is guaranteed to be satisfiable (if a satisfiable assignment exists).

Crucially, this algorithm exploits the monotonicity of the PSTL formula to guarantee that there is at most one point during the bisection search where the satisfaction of φ can

³We postpone any discussion of how to choose such a scalarization to future work.

⁴For simplicity, we have omitted a number of optimizations in Algorithm 1. For example, at the beginning one can check if ν^ℓ is a non-satisfying assignment and terminate early due to monotonicity. We note that this does not change the worst case run time nor the worst case error.

Algorithm 1: Iterated Binary Search to compute $\pi_{\text{lex}}(\mathbf{x})$

Input: $\mathbf{x}(t), \varphi(\mathbf{p}), \mathcal{P}, \mathcal{D}_{\mathcal{P}}, (j_1, \dots, j_n), \bar{\epsilon} > 0, \preceq_{\text{lex}}$
Result: $\pi_{\text{lex}}(\mathbf{x})$

```

1  $\nu^\ell(\mathbf{p}) \leftarrow \perp; \nu^u(\mathbf{p}) \leftarrow \top; \nu \leftarrow \perp$ 
2 for  $i = 1$  to  $n$  do
3   while  $|\nu^u(p_{j_i}) - \nu^\ell(p_{j_i})| > \epsilon_i$  do
4      $\nu(p_{j_i}) \leftarrow \frac{1}{2} (\nu^\ell(p_{j_i}) + \nu^u(p_{j_i}))$ 
5     if  $\mathbf{x} \models \nu(\mathbf{p})$  then  $\nu^u(\mathbf{p}) \leftarrow \nu(\mathbf{p})$  else  $\nu^\ell(\mathbf{p}) \leftarrow \nu(\mathbf{p})$ 
6 return  $\pi_{\text{lex}}(\mathbf{x}) \leftarrow \nu^\ell(\mathbf{p})$ 

```

change. The number of iterations for each parameter index i is bounded above by $\log \left[\frac{\sup(\mathcal{D}_{\mathcal{P}_i}) - \inf(\mathcal{D}_{\mathcal{P}_i})}{\epsilon_i} \right]$, and the number of parameters. This gives us an algorithm with complexity that grows linearly in the number of parameters and logarithmically in the desired precision.

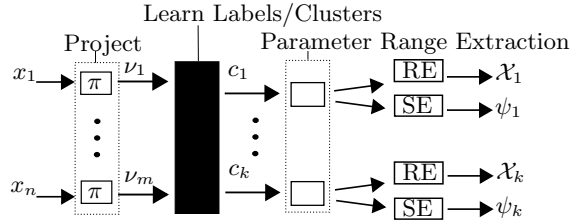


Figure 1: Overview of the temporal projection based learning process. We first project each trace in the sample set to the parameter space and then apply a black-box clustering scheme. For each cluster, we then obtain the bounding hyperbox for the parameter valuations in that cluster. This is followed by Representative Extraction (RE) and STL Extraction (SE) to get a set of representatives and an STL artifact respectively.

4. CLUSTERING AND STL EXTRACTION

Rationale for Temporal Projection. In the previous section, we defined projection functions to map timed traces to valuations in the parameter space. In this section, we interpret the results of projection as a feature space on which to apply classical clustering algorithms. We focus on features defined by PSTL formulas, because in some cases, there is an obvious visual relation between parameter valuations of the PSTL formula and the shape of a timed trace. We elaborate with an example.

Example 2. Consider the following PSTL template:

$$\varphi_{h,w} = \mathbf{F}(x > h \wedge \mathbf{F}_{[0,w]}(x = 0)) \quad (4.1)$$

using the lexicographic ordering $\preceq_{\text{lex}}$ that uses the order (h, w) for the parameters h and w . Next consider the following parameterized class of traces, $X_s = \{f((h, w), t) : \forall h, w \in (1, 10)\}$, where f is given in Eq. 4.2.

$$f((h, w), t) = \begin{cases} \frac{h}{w}t & t \in [0, w] \\ h - \frac{h}{w}(t - w) & t \in (w, 2w] \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

Refer to Fig. 4 for examples of timed traces in X_s . Note that this set as been carefully crafted to have height at most

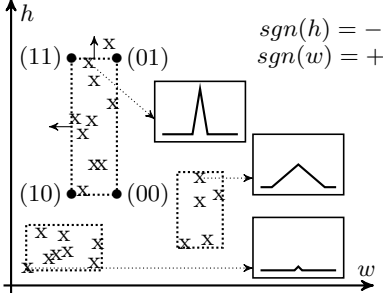


Figure 2: Illustration hypothetical three cluster distributions and examples of spike instances.

h and width at most $2w$. The lexicographic projection then recovers the h and w of a given spike. Now suppose that the distribution on X_{spike} has 3 sub-populations. For example, the first might be “spikes” characterized by “large” h and “small” w . Next we might have a “small spike” sub-population. In this population h and w are expected to be small. Finally, perhaps we have a third population, “flat” corresponding to small h and large w . An illustration of this hypothetical distribution on X_{spike} is given in Fig. 2.

In the above example, we can see that parameter valuations that are nearby correspond to timed traces that have similar shapes. Furthermore, there is a clear separation between clusters representing parameter valuations corresponding to trace-projections for traces with different shapes. This means that if we can leverage traditional machine learning algorithms for clustering, we can group timed traces that have similar shapes (modulo the chosen PSTL formulas) together.

In our case studies we primarily leverage two unsupervised algorithms implemented in the `scikit-learn` toolkit [9]: 1. Gaussian Mixture Models (GMMs) 2. One Class SVM (support vector machine) with a Gaussian Kernel. Gaussians (in both algorithms) appear as natural candidates because bounding hyperboxes for clusters in the parameter space are typically not overlapping, and the our examples have relatively low dimension. This means that the Euclidean distance between parameter valuations serves as a similarity metric (as it bounds the equiprobable sphere around the Gaussian). Because we believe that the data set contains a number of distinct sub-populations, we employ mixture models. These enable explaining data that is drawn from the union/mixture of multiple sub-populations (that could for example result in overlapping Gaussians). If we believe that there is a primary population with a small number of anomalies, we employ a one class SVM. Such a classifier is designed to fit the primary population using a Gaussian and identify outliers.

4.1 Hyperbox Clusters

Rationale for choosing clusters shaped as hyperboxes. We recall that each valuation in the parameter space corresponds to an STL formula. Thus, a hyperbox in the parameter space, in general, corresponds to an infinite set of STL formulas. However, for monotonic PSTL formulas, as the validity domain is upward closed, we show that a hyperbox in the parameter space actually represents a single STL formula. In fact, the vertex representing the infimal valuation in the hyperbox w.r.t. the $\leq$ order corresponds to the least permissive STL formula, the supremal valuation corresponds to the most permissive STL formula, and the remaining vertices of the hyperbox correspond to incomparable STL formulas. As we shall

see in Sec. 4.2, this observation allows us to get an STL formula of length bounded by the number of dimensions in the parameter space.

Hence, after running the clustering algorithms, for each cluster we compute the smallest axis aligned hyperbox that contains the entire cluster. We remark that this is a simple search for the min and max along each dimension. In rest of the section, we assume that clusters are represented by enclosing hyperboxes⁵. Furthermore, for certain arrangements of the clusters in the parameter space, the formula corresponding to a cluster can be simplified further. We now formalize these notions.

We denote by $\mathcal{H}$ the infinite set of all axis aligned hyperboxes in $\mathcal{D}_{\mathcal{P}}$. The set $\mathcal{H}$ has the following properties. Box $B \in \mathcal{H}$ is compactly parameterized by 2 valuations $\nu_{\perp}$ and $\nu_{\top}$, where $\nu_{\perp}$ is the infimal vertex of the box and $\nu_{\top}$ is supremal vertex. We denote by $B(\nu_{\perp}, \nu_{\top})$ the hyperbox with points $\nu_{\perp}$ and $\nu_{\top}$ as extremal vertices.

DEFINITION 4.1 (UPWARD CLOSURE OF A VALUATION). For a valuation ν , its upward closure (denoted $U(\nu)$) is the set $\{\nu' \mid \nu \leq \nu'\}$.

Note that we can now define a hyperbox B as $U(\nu_{\perp}) \setminus \bigcup_i U(\nu_i)$, where ν_i are the valuations corresponding to the vertices of the box B except $\nu_{\perp}$. It follows that the hyperbox is open on the faces not connected to $\nu_{\perp}$.

4.2 STL Extraction

Let ν be an arbitrary valuation in a hyperbox B . Further, let $\varphi_{\perp} \equiv \varphi(\nu_{\perp})$, and let φ_i be the formula corresponding to the i^{th} vertex of B that is not $\nu_{\perp}$. Then, the following properties hold for any trace x :

- If $x \models \varphi(\nu(\mathbf{p}))$ then $x \models \varphi_{\perp}$, and,
- If $x \models \varphi(\nu(\mathbf{p}))$ then $x \not\models \varphi_i$.

In other words, any valuation $\nu \in B$ satisfies the following:

$$\varphi_B = \varphi_{\perp} \wedge \left(\bigwedge_{i=1}^{2^{|\mathcal{P}|-1}} \neg \varphi_i \right) \quad (4.3)$$

In fact, due to the product order on the hyperbox indices, only the vertices corresponding to powers of 2 are required to encode the hyperbox (due to the upward closure), thus bounding the formula length to $|\mathcal{P}| + 1$ and φ_B is equivalent to the following formula:

$$\varphi_B = \varphi_{\perp} \wedge \left(\bigwedge_{i=1}^{|\mathcal{P}|} \neg \varphi_{2^i} \right) \quad (4.4)$$

Generalizing concepts. In some clustering scenarios, a hyperbox concept can be too restrictive. For a given hyperbox-cluster, the upward closure of some of its vertices does not contain any other cluster, then we can relax the hyperbox concept. This relaxation comes with an advantage: STL formulas corresponding to relaxed concepts are smaller and thus simpler. For example, a concept bounding a spike is naturally unbounded in the slope/magnitude. Furthermore, as we shall see, unconstraining a hyperbox boundary simplifies the learnt STL making the learnt formula more accessible. As such it behooves us to generalize our concept class. We seek generalizations that maintain the following properties:

⁵In general the hyperboxes that we may obtain may not be disjoint. In this case, we can cover the points in the cluster with a minimal number of disjoint hyperboxes. We omit these details for simplicity.

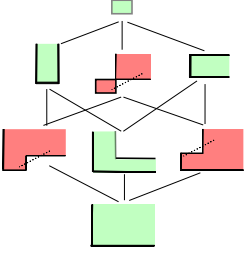


Figure 3: Illustration of the various shapes that are generated by taking the greatest lower bound of the constraints in 2-d. It also demonstrates the disallowed shapes due to the existence of points that are comparable, but whose convex combination it outside the concept.

- The learned concept corresponds to an STL formula that is a simple Boolean combination of instantiations of the given PSTL formula.
- Between any two valuations in the concept, the line segment between the valuations is also in the concept. This helps cognition of the concept using boundary cases and allows “cognitive interpolation” modulo the lens of the PSTL template.

We begin by formalizing the requirement of being able to perform cognitive interpolation between valuations in the concept.

DEFINITION 4.2 (COMPARABLE CONVEXITY).

$C \subset \mathcal{D}_{\mathcal{P}}$ is comparably convex iff $\forall \nu, \nu' \in C$:

$$\nu < \nu' \implies \forall \lambda \in [0, 1], (\nu + \lambda(\nu' - \nu)) \in C \quad (4.5)$$

This definition asserts that the line between any two comparable valuations in the concept be contained in the concept (as is clearly the case with hyperboxes). In figure 3, we illustrate the possible generalizations of the concept class that can be generated using unions and set subtractions of the upward closures of the hyperbox vertices. We indicate undesirable concepts (in red) via a dotted line witnessing the non-comparable convexity. In the next theorem we demonstrate that if such a concept C is comparably convex, then C can be characterized using a conjunction of the hyperbox vertices. This is desirable since it limits the complexity of the learnt STL.

THEOREM 4.1. *Let C be a region defined as some Boolean combination of $U(\nu)$ and $\overline{U(\nu)}$, where ν is a vertex of a hyperbox B . If $C \supseteq B$ and C is comparably convex, then the concept C can be represented by a formula φ_C that is a conjunction of STL formulas, that contains $\varphi_{\perp}$, and all other conjuncts are STL formulas corresponding to vertices of B or their negations.*

PROOF. For brevity, we provide a proof sketch (with a proof provided in the appendix). We observe that as $B \subseteq C$, we must include $\nu_{\perp}$ in C , and thus φ_C must contain $\varphi_{\perp}$ as a conjunct. Further, we claim that for C to be comparably convex, we cannot allow arbitrary disjunctions over the $U(\nu)$, $\overline{U(\nu)}$ sets. Further the upward closure in the parameter space implies that if we include a vertex ν , then any vertex in $U(\nu)$ that lies on a connecting face must also be included. A given conjunction of formulas corresponding to vertices (or their negations) indicates whether or not to include vertices in their respective upward closures. The proof exploits this by showing that if the concept is comparably convex along a face, then there exists a conjunction with the formula (or its negation) for the corresponding vertex included. ■

THEOREM 4.2. *The size of φ_C is at most $(|\mathcal{P}|+1)(|\varphi|+1)$.*

PROOF. We observe that for the concept corresponding to the box B , the length of the formula is $(|\mathcal{P}|+1)(|\varphi|+1)$ (from Eq. 4.4). We then claim that for any other concept C , the formula is shorter in length. We first observe that as we have $f_{\perp}$ as a conjunct in φ_C , we cannot have any the formula for any other vertex appear positively in φ_C (because of upward closure). Any other formula, would necessarily be shorter than φ_B , as φ_B contains negations with respect to formulas for all other vertices. ■

5. AUDITABILITY/REPRESENTATIVE EXTRACTION

One may argue that the existence of an STL formula corresponding to the learnt parameter region, while nice is difficult to interpret for practicing engineers. Rather, it is often more useful to understand by example the typical and extremal behaviors in the learnt region. Because the valuations in a concept are monotonically ordered w.r.t. satisfaction the traces that project closest to $\nu_{\perp}$ and $\nu_{\top}$ are natural candidate examples (as well as the trace closest to the center). In the step detection and overshoot case studies we will see how one can cognitively interpolate and understand what traces lie between the edge cases.

In this section we attempt to formalize why observing the extremal examples of the learned concept yields unreasonable confidence in the concept. We choose to base this formalism around the notion of a homotopy lifting property[10]. Specifically, we seek 2 homotopy liftings: first, a map $\hat{\pi}$ s.t. for the projection π and any curve $\theta(\lambda)$ in $\mathcal{D}_{\mathcal{P}}$, $\pi(\hat{\pi}(\theta(\lambda))) = \theta(\lambda)$ and $\hat{\pi}(\theta(\lambda))$ is a continuous deformation. This first lifting enables the cognitive interpolation between any two valuations. The second lifting requires that there is a lifting of the continuous deformations between curves in $\mathcal{D}_{\mathcal{P}}$ to a continuous deformations of the corresponding homotopies in X . The intuition is that when interpolating between examples we are finding a continuous deformation in X that projects to a line in $\mathcal{D}_{\mathcal{P}}$. One imagines an engineer exploring a concept by adjusting sliders corresponding to parameter values within the concept. These adjustments correspond to a path in $\mathcal{D}_{\mathcal{P}}$ and finding an example that projects to that point would correspond to the homotopy lifting. We refer to the engineer’s ability to sensibly/systematically explore the concept as “human auditability”. The existence of such liftings would imply that one can simply examine the extremal cases and be confident there are no “holes” or discontinuities in your cognitive interpolation between points.⁶

Example 3. Consider again the spike template $\varphi_{h,w}$, 4.1 restricted again to X_{spike} . Further, let φ_B encode the bounding box. Define $\psi \triangleq \varphi \wedge \varphi_B$. Notice that ψ is identical in behavior as $\varphi_{h,w}$, but is bound to the extracted parameter region. Observe that any curve between any two parameters in the bounding box can be lifted to a homotopy in the signal space s.t. there exists a projection map $\pi \in \Pi_{\psi}$ back to the line. Namely, fix π as lexicographic order $h \preceq_{\text{lex}} w$ and take any curve in the parameter space $\theta : [0, 1] \rightarrow \mathcal{D}_{\mathcal{P}}$. Recalling that the lexicographic ordering exactly recovers w and h , a valid homotopy lifting of $\theta(\lambda)$ into X is:

$$g(t, \lambda) = f(\theta(\lambda), t) \quad (5.1)$$

⁶In addition to helping with human auditability, one might consider using the learnt homotopies as an environment abstraction for model checking/falsification.

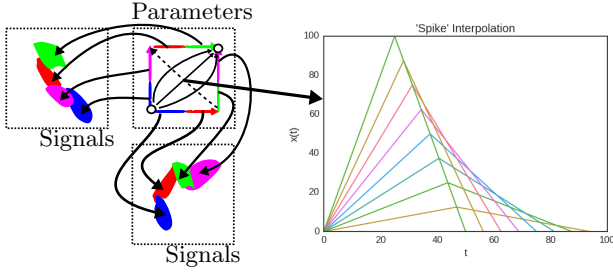


Figure 4: Illustrates 2 paths in parameter space being mapped to a homotopies in the signal space. On the right we give an explicit sampling of a lifting of the diagonal in the spike template.

Recall that without our lifting condition, any 2 continuous curves between two points in the parameter space are in the same homotopy class (inherited from $\mathbb{R}^{|\theta|}$). Also notice that there are no curves in the bounding hyperbox that cannot be lifted. Thus, we can continuously deform both the curve in the parameter space and the homotopy in X_{spike} .

We conjecture the following to be true for any learnt concept, and leave them open.

1. Between any two points in a comparably convex concept, there is a monotonically increasing curve that can be lifted to the signal space.
2. Between any 2 monotonic curves in the parameter region, there exists a homotopy between them (in the set on monotonic curves) that satisfies the lifting property.
3. Between all the Manhattan Paths and the box diagonal in the parameter space, there exists a homotopy that can be lifted.

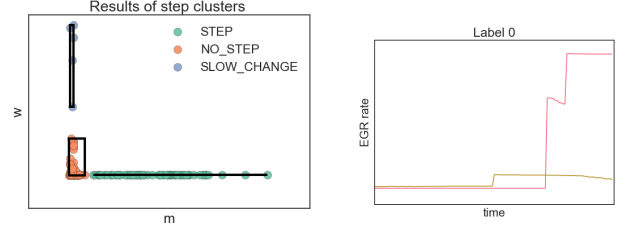
6. CASE STUDIES

Implementation Details. We utilize the Breach tool [8] and a combination of Matlab/Python scripts to synthesis the tightest parameter for a single trace. Breach supports monotonic parameter synthesis via lexicographic ordering as well as general cost functions. We heavily leverage the vast collection of unsupervised learning algorithms in the `scikit-learn` toolkit [9] to move between projection and parameter region extraction.

6.1 Diesel Engine

We examine timed traces for the Exhaust Gas Recirculation (EGR) rate obtained from an early prototype of a Diesel Engine airpath control problem. The controller sets a reference signal for the EGR and the system attempts to match that rate. Engineers typically test such systems by visually inspecting the response to a step or impulse. The engineer’s mental model is that the system should respond to the references in a timely manner, not overshoot too much, and to avoid “spiky” behavior. The ST-Lib library [11] has a pool of PSTL formulas designed to detect violations of such properties. A primary difficulty in using ST-Lib is finding the correct set of parameters to characterize undesired behavior. In the following case study, we show how to do so in an unsupervised fashion.

Reference Step Detection. Many ST-Lib formulas are “step-triggered”, *i.e.*, they are of the form: $\mathbf{F}(\text{step} \wedge \phi)$. Thus automatically characterizing steps defines a fundamental specification primitive. In particular the **rising-step**



(a) Parameter Space

(b) Step Representatives

Figure 5: a) Results of projecting a 1 second sliding window of the EGR rate through the step PSTL template 6.1. The direction of upward closure as specified by the parameter polarities is to the bottom right. Observe that there are in fact 3 distinctive clusters. b) Representatives for each cluster generally match expectation, expect perhaps the step-like curve labeled slow_change. Given the time scales involved, we decided not to consider this to be a step.

template used is:

$$\text{step}_{(m,w)} \triangleq \mathbf{F}(\ddot{x} > m \wedge \mathbf{F}_{[0,w]}(\ddot{x} < -m)) \quad (6.1)$$

This requirement first reduces step detection (via a discrete-time derivative) to spike detection and then applies the ST-Lib spike detection template that introduces a second discrete-time derivative. As the view of PSTL is signal-centric, such operations can be introduced as new timed traces of a new signal, and do not require any modification of the logic.

Expected Behavior. Intuitively we expect to see 3 types of behaviors:

1. Step/Spike: The slope will be large and there will be a short width: $\max_{(m,-w)}$
2. Slow Change: **Slow** long term rises in the reference signal. Thus the width should be large $\max_w$
3. Flat/No-Step: There is no rise in the signal. (note, there may be a falling step): $\min_{(m,w)}$

Thus, we expect to see 3 clusters. A small Euclidean distance between two valuations is a good indicator of membership of both the valuations to the same cluster. Hence, we choose to cluster using Gaussian Mixture Models.

For this experiment we had 33 experimental time series of variable length. As a preprocessing step, we used a sliding window with a window size of 1 second and a sliding offset of 0.5 seconds to generate equal length traces. The sliding window size and the offset size was chosen by observation and experience to capture the significant local behaviors. In general, such a selection could be automated based on statistical criteria. Each trace generated is then preprocessed by numerically computing the second derivative⁷. Next we project to the parameter space using Algorithm 1 as implemented in Breach. The lexicographic ordering used was (m, w) , where m is optimized first followed by w . We then normalized the parameters to lie between 0, 1 and applied a 3-cluster GMM to generate labels, compute bounding boxes, the representative STL formula, and representative examples for each cluster.

⁷Here, because the derivative operator can introduce considerable noise into data, care should be taken when working with noisy data to consider not just the neighboring points, but also the entire neighborhood to compute a smoothed derivative. We remark that the discrete-time derivative can often be approximated by a noise-robust operation (such as the difference from a rolling mean or median

Results. As we can see from Fig. 5a, our 3 cluster hypothesis is confirmed. The STL formula for the “step” cluster is:

$$\text{step} \triangleq \mathbf{F}(\ddot{x} > m^* \wedge \mathbf{F}_{[0, w^*]}(\ddot{x} < -m^*)) \quad (6.2)$$

Here, the values m^* and w^* are suppressed for proprietary reasons. We choose to focus on the step cluster as it will be used as an artifact in the next experiment⁸. The traces corresponding to the infimal point and the suprenal point of the tightest bounding box are shown in figure 5b. The results match expectations, with one trace exhibiting a small step and the other exhibiting a much larger step. The omitted representatives are as expected mostly flat, gradually changing over time, or falling steps (which are not captured in 6.1 due to a restriction to $c \geq 0$).

Reference Overshoot Detection. With step classification done and an STL formula extracted, we would now like to detect overshoots w.r.t. a reference signal following a step in the reference signal. To capture overshoot behavior we use the following PSTL formula:

$$\varphi_{\text{overshoot}}(c, w) \triangleq \text{step} \wedge \mathbf{F}_{[0, w]}(x - x_r) > c \quad (6.3)$$

Here, step is the previously obtained step formula with x_r instead of x , and $c, w \in \mathbb{R}^+$

Expected Behavior. Qualitatively we expect 3 types of behavior. First and foremost, we predict that some of the subsequences will exhibit overshoot characterized by high c . Conversely, if the system properly tracks the reference, we expect to see a very small c . Lastly, since we limited since c to take values in $\mathbb{R}^+$, failure to rise above the reference will result in a valuation with $c = 0$ and $w = \top$ (since we have $c \preceq_{\text{lex}} w$).

Because we expect one or two linear modes to dominate the post step behavior, we expect to potentially have some oscillatory behavior about the reference. Given the time scales studied, we expected up to two crossings for a major overshoot. Thus we expect 2 sub-populations during clustering, corresponding to the first and 2nd intersection. Given 2 types of overshoots at 2 times, plus a cluster for no rise gives a total of 5 expected clusters. Again, euclidean distance appears to be a good measure of relatedness, so we chose to use a GMM.

Experiment. For this experiment reuse the same data-set as the step experiment. Rather than computing a sliding window, we use Breach’s robustness evaluation to compute when the template is matched. For each matching time point, we generate a trace corresponding to the next second. The parameter synthesis and data analysis are analogous to the step experiment, this time having 5 instead of 3 clusters.

Results. As expected we identify 5 clusters. The top-left cluster corresponds to overshoot and the top right cluster corresponds to a steady state error/2nd late overshoot. The relaxed-formula for the overshoot cluster is: Using relaxation based on upward closure, we can fully relax the hyperbox since there were not conflicting clusters (has a non-empty intersection with the upward closure of an enclosing-hyperbox-corner) and qualitatively a larger overshoot is still an overshoot.

$$\varphi_{\text{overshoot}} \triangleq \mathbf{F}_{[0, w^*]}(x > c^*) \quad (6.4)$$

About overlapping clusters. Notice that the overshoot cluster intersects (minimally) with the normal operation cluster. Inspecting the corresponding minima and maximal points,

⁸and of course for space/brevity/etc

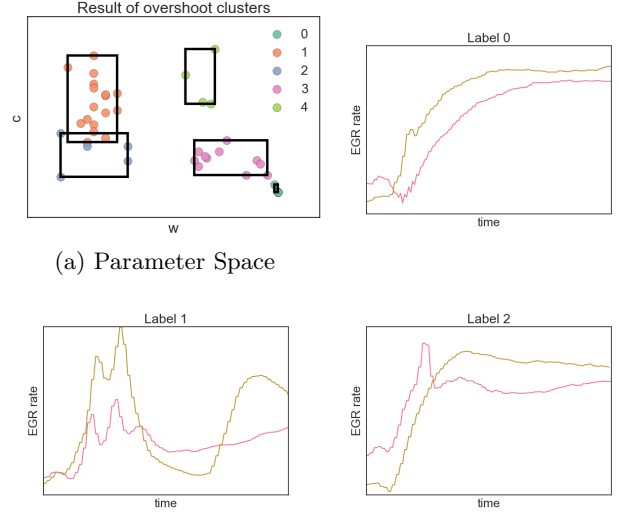


Figure 6: a) Results of fitting and clustering the EGR data w.r.t. the overshoot template. Note that there are in fact 5 clusters. We do not consider the overlapping clusters to be a problem in the case since the threshold between overshoot and not overshoot is ambiguous. b) Edge case examples from a subset of the clusters.

these are indeed ambiguous. This is where the quantitative semantics of the STL formula can be used to give a confidence in the detection of an anomaly. This also emphasizes the fallacy that there exists a hard cutoff between overshoot and normal behavior. In practice engineers understand that no such cutoff exists and that buffers are required in order to ensure robustness. Thus the Boolean semantics are not accurate till the quantitative robustness is “large”. Here we interpret this to mean that the minimal overlapping of the hypercubes is acceptable and no further refinement is actually needed.

6.2 Traffic Behavior on the US-101 highway

In order to nourish research in modeling and predicting driver behaviors, the next generation simulation computer program collected detailed traffic data on southbound US-101 freeway, in Los Angeles [7]. The pre-selected segment of the freeway is about 640 meters in length and consists of five main lanes and some auxiliary lanes. Traffic through the segment was monitored and recorded through eight synchronized cameras, mounted on top of the buildings next to the freeway. A total of 45 minutes of traffic data was recorded including vehicle trajectory data providing lane positions of each vehicle within the study area. Through the trajectory data, we can classify the aggressiveness of driving behavior by checking how often a driver switches lanes and the dwell time in each lane before switching. As lane 1 is the rightmost lane, lane 5 is the leftmost lane, and auxiliary lanes are mainly for entering and exiting the freeway, in these lanes, lane-switching behavior is necessary. Thus, we focus our analysis on the lane switching behaviors on lanes 2, 3, and 4. Each vehicle trajectory $x(t)$, stores the lane position for the vehicle, and we use the following STL formula to capture the dwell time in L_i before switching to another lane:

$$\mathbf{F}(x \neq L_i \wedge (\mathbf{F}_{[0, \epsilon]} x = L_i \mathbf{U}_{[\epsilon, \tau_i]} x \neq L_i)) \quad (6.5)$$

Expected Behavior. Based on the driving behavior at different lanes, we expect four types of behavior:

C	τ_2	τ_3	τ_4	T	Traj.]
C0	$\top$	$\top$	$\top$	T1	626
C1	$\top$	$\top$	$[\perp, 77.68]$	T2	115
C2	$[\perp, 66.97]$	$\top$	$\top$	T2	44
C3	$\top$	$[0.26, 30.29]$	$[\perp, 65.35]$	T3	52
C4	$\top$	$[\perp, 70.17]$	$\top$	T2	32
C5	$[0.56, 54.00]$	$[\perp, 39.75]$	$\top$	T3	14
C6	$[0.56, 32.00]$	$[1.76, 24.28]$	$[\perp, 30.96]$	T4	12
C7	75.77	$\top$	$\top$	T1	1

Figure 7: The GMM clustering results of the lane switching behavior analysis. Table includes the cluster label, parameter values, the type of the expected behavior for each cluster, and the number of trajectories in the cluster.

- T1. Conservative driving behavior: only switching to adjacent lanes from L_i and never changing back to L_i , for all $i \in \{2, 3, 4\}$. This corresponds to “no weaving.”
T2. Normal driving behavior, from L_i switching to adjacent lanes and coming back to overtake a slow vehicle in front, where $i \in \{2, 3, 4\}$
T3. Slightly aggressive behavior, weaving between 2 lanes.
T4. Aggressive behavior, weaving between three lanes.

Results. For this experiment, from 4824 total vehicle trajectories, we discard trajectories with no lane switching behavior and group them with the conservative driving behaviors. We analyze the remaining 896 targeted trajectories that have at least one lane-switch behavior, and each trajectory is at most 100 seconds long. We project each trajectory to a 3D parameter space by synthesizing tight bounds for parameters τ_2 , τ_3 , and τ_4 of the STL formula 6.5 using Breach. As all parameters are independent, lexicographic ordering has no impact on the final clustering results. After normalizing the parameters by centering and scaling, we apply GMM-based clustering to generate bounding hyperboxes and the associated STL formulas.

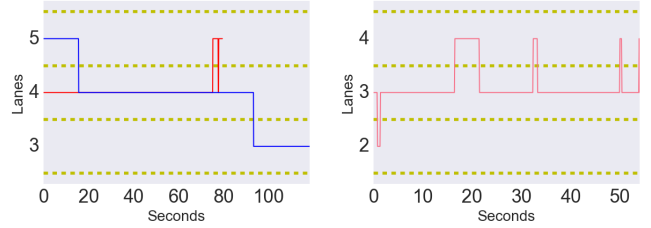
The resulting clusters are shown in Table 7. The table lists all GMM clusters, the corresponding parameter values, the expected type that the cluster belongs to, and the number of trajectories in this cluster. The largest cluster, Cluster 0, contains behaviors without any weaving behavior. This matches the expectation that the majority of drivers are (thankfully!) not aggressive.

We observe that GMM clustering results in different combinations of lane behaviors as lane behaviors are independent from each other. For instance, Cluster 1, 2, 4 individually map to the weaving behavior solely on Lanes 4, 2, and 3. Cluster 3 and 5 represent the weaving behavior involving 2 lanes. Cluster 6 represents aggressive behavior and one of the representative is shown in Figure 8b. We consider Cluster 7 as an anomaly for Cluster 2 as it has only 1 trajectory.

For clusters 1, 2, and 4, we cannot distinguish the if drivers were rapidly weaving or weaving within a short period of time, due to the scarcity of the data. However, as shown in Figure 8a, for cluster 1, there were two different behaviors; one involving rapid lane-switching (red trace), one where the driver switched lanes more slowly (blue trace). We then apply a one class SVM to further explore the internal structure of Cluster 1 to distinguish between aggressive and normal behaviors.

6.3 CPS Grader

In [14], the authors characterize the validity domain of monotonic PSTL based on labeled traces in order to create a classifier for whether simulations of student solutions



(a) Examples from Cluster 1 (b) Aggressive Behavior

Figure 8: 8a shows the external cases in cluster 1. Both the blue and the red trace enter into lane 4 and then leave within 77.68 seconds. Upon the inspection, these two extremal cases are qualitatively very different. The blue car moves from lane 5 to lane 4, remains for about 60sec and then moves to lane 3. The red car appears to use lane 5 to pass another car, move’s back into lane 4 and then to lane 3 shortly after. These situations are characterized by very different τ_4 values. Inspecting the data, most of cluster 1 has a large τ_4 value. As such we chose to subdivide the behavior further using a one class svm and interpreted the small values of τ_4 to be more “aggressive”. An example is given in 8b.

were acceptable The tests in [14] involved the simulation of an IRobot Create and a student generated controller. The controller needed to navigate the robot up an incline and around static obstacles. To test this, the authors created a series of parameterized environments and a set of PSTL formula that characterized failure. In this work we attempt to reproduce a somewhat arbitrary subset of the results shown in [14] that required no additional preprocessing.

Obstacle Avoidance. We focus on 2 tests centered around obstacle avoidance. The authors used an environment where an obstacle is placed in front of a moving robot and the robot is expected to bypass the obstacle and reorient to it’s pre-collision orientation before continuing. The relevant PSTL formulas are:

1. Failing simple obstacle avoidance:

$$\varphi_{avoid}(\tau, y_{min}) = \mathbf{G}_{[0, \tau]}(pos.y < y_{min}) \quad (6.6)$$

2. Failing re-orienting after obstacle avoidance:

$$\varphi_{reorient} = \mathbf{G}_{[0, \tau]}(pos.y < y_{min} \vee pos.x > x_{max}) \quad (6.7)$$

Results. The primary observation in both case studies is that the vast majority of data is captured in a relatively small parameter range. Upon investigation, it was revealed the students were able to submit multiple solutions for grading - each corresponding to a trace. This biases the dataset towards incorrect solutions since one expects the student to produce many incorrect solutions and then a few final correct solutions. Further, as discussed in the caption for figure 9, the results imply that a classifier for label 0 would have a low misclassification rate. A similar result follows from 10 when given the information that the initial orientation is 45° implying increasing x and y . Thus the bottom cluster corresponds to failure to pass the obstacle with clusters 2 and 3 corresponding to failures to reorient.

Conclusion. In this paper, we explore an unsupervised learning technique that takes as input a set of

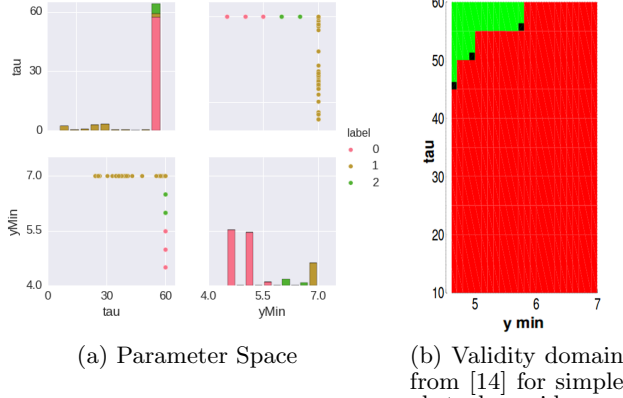


Figure 9: 9a is the pairwise distribution plot of the components learnt via a 3 component GMM for the simple obstacle avoidance test. Observe that over 95% of the examples are captured by cluster 0. Focusing on the top right figure in 9a, one can see that cluster 0 corresponds almost exactly with the upper boundary of the validity domain in figure 9b. This region corresponds to the classifier used to determine if a student’s solution was correct or incorrect. Thus, we note that the extracted STL for classifying cluster 0 performs comparably with the original results.

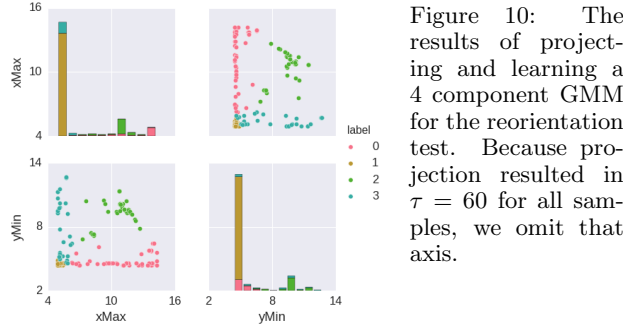


Figure 10: The results of projecting and learning a 4 component GMM for the reorientation test. Because projection resulted in $\tau = 60$ for all samples, we omit that axis.

timed traces and produces a set of disjoint clusters, each containing a subset of the given timed traces. Furthermore, each cluster C is associated with a Signal Temporal Logic formula φ_C , such that all traces in C satisfy φ_C . The key step is to project the given timed trace into the space generated by the parameters of a selected Parametric STL formula. We define a notion of auditability based on the mathematical idea of homotopy equivalence, and conjecture that the clustering algorithm we propose and the STL formulas that we learn guarantee auditable concept classes.

Future Work. For future work, we will study extensions of this approach to supervised, semi-supervised, and active learning. A key missing component in this work is a principled way to select a projection function (perhaps via learning or posterior methods). Other possible extensions involve integration with systematic PSTL enumeration, learning non-monotonic PSTL formula, and developing applications/theory around the demonstrated homotopy lifting.

7. REFERENCES

- [1] H. Abbas, K. J. Jiang, Z. Jiang, and R. Mangharam. Towards Model Checking of Implantable Cardioverter Defibrillators. In *Proc. of HSCC*, pages 87–92, 2016.
- [2] E. Ackerman. Google’s Autonomous Cars Are Smarter Than Ever at 700,000 Miles. *IEEE Spectrum*, 2014.
- [3] R. Alur and T. A. Henzinger. A really temporal logic. *JACM*, 41(1):181–203, 1994.
- [4] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In *Proc. of RV*, pages 147–160, 2011.
- [5] E. Bartocci, L. Bortolussi, and G. Sanguinetti. Learning temporal logical properties discriminating ECG models of cardiac arrhythmias. *arXiv preprint arXiv:1312.7523*, 2013.
- [6] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *Proc. of HSCC*, pages 1–10, 2016.
- [7] J. Colyar and J. Halkias. US highway 101 dataset. *Federal Highway Administration (FHWA)*, *Tech. Rep. FHWA-HRT-07-030*, 2007.
- [8] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Proc. of CAV*, pages 167–170, 2010.
- [9] F. Pedregosa, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [10] A. Hatcher. *Algebraic Topology*. Cambridge Univ. Press, 2000.
- [11] J. Kapinski et al. ST-Lib: A Library for Specifying and Classifying Model Behaviors. In *SAE Technical Paper*. SAE, 2016.
- [12] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *IEEE TCAD of ICS*, 34(11):1704–1717, 2015.
- [13] A. Jones, Z. Kong, and C. Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *Proc. of CDC*, pages 848–853, 2014.
- [14] G. Juniwal, A. Donzé, J. C. Jensen, and S. A. Seshia. CPSGrader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory. In *Proc. of EMSOFT*, page 24, 2014.
- [15] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.
- [16] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for data mining applications. In *Proc. of KDD*, pages 285–289, 2000.
- [17] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta. Temporal logic inference for classification and prediction from data. In *Proc. of HSCC*, pages 273–282, 2014.
- [18] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.
- [19] J. Legriel, C. Le Guernic, S. Cotton, and O. Maler. Approximating the pareto front of multi-criteria optimization problems. In *Proc. of TACAS*, pages 69–83, 2010.
- [20] T. W. Liao. Clustering of time series data survey. *Pattern recognition*, 38(11):1857–1874, 2005.
- [21] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS/FTRTFT*, pages 152–166, 2004.
- [22] K. Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

8. APPENDIX

8.1 Proof that formula in Eq. 2.5 is not monotonic

PROPOSITION 8.1. *The following formula is not monotonic:*

$$\mathbf{F}((x \leq h) \wedge \mathbf{F}_{[0,w]}(x \geq h)).$$

PROOF. Note that for φ specified in (2.5), $\mathbf{p} = (h, w)$. We will show that $\varphi(\mathbf{p})$ is neither monotonically increasing nor decreasing. Consider two valuations ν and ν' such that $\nu(w) = \nu'(w)$, and $\nu(h) \neq \nu'(h)$.

Case 1: (Not monotonically increasing) Suppose, for some $c \in \mathbb{R}$, $\nu(h) = c$, $\nu'(h) = c'$, and $c < c'$. Suppose that for all t $x(t) = c$, clearly, $x \models \varphi(\nu(\mathbf{p}))$. Furthermore, as $c < c'$ and $\forall t : x(t) = c$, $(x, t) \models (x \leq \nu'(h))$. However, there is no t such that $x(t) > c'$, i.e., $x \not\models \varphi(\nu'(\mathbf{p}))$. In other words, φ is not monotonically increasing in h .

Case 2: (Not monotonically decreasing) Now consider valuations $\nu(h) = c$, $\nu'(h) = c''$, and $c' < c$. Again, let $x(t) = c$, for all t . As before, $x \models \varphi(\nu(\mathbf{p}))$. However, there is no t where $x \leq c''$, thus $x \not\models \varphi(\nu'(\mathbf{p}))$. Thus, x is not monotonically decreasing.

Therefore φ given in Eq. (2.5) is not monotonic. ■

8.2 Proof of Theorem 4.1

THEOREM 8.1. *Given a hyperbox B , let C be a region defined as some Boolean combination of $U(\nu)$ and $\bar{U}(\nu)$, where ν is a vertex of B . If $C \supseteq B$ and C is comparably convex, then the concept C can be represented by a formula φ_C that is a conjunction of STL formulas, that contains $\varphi_\perp$, and all other conjuncts are STL formulas corresponding to vertices of B or their negations.*

PROOF. We begin by considering the contrapositive: C is not a conjunction of the vertices of the box (and their negations) $\implies C$ is not comparably convex or $B \not\subseteq C$

1. If φ_C is not a conjunction, then it can be written in Disjunctive Normal Form as a disjunction of prime implicants $\varphi_C = c_1 \vee (c_2 \vee \dots) \triangleq c_1 \vee C'$ ⁹

2. Case 1: $\forall c_i, \neg(\varphi_B \implies c_i)$

Because the box corners are upward closed, $c_i \implies \varphi_\perp$ iff c_i includes the hyperbox (since the other corners control upward closures outside the hyperbox). Thus, none of the c_i contain $\varphi_\perp$. Therefore, $\neg(\varphi_B \implies \varphi_C)$, and thus $B \not\subseteq C$ ¹⁰ ■

3. Case 2: $\exists c_i$ s.t. $\varphi_B \implies c_i$

W.L.O.G. assume c_i is c_1 . Therefore, $\nu_\perp$ is in the region induced by c_1 . Because c_1 is a prime implicant not included in C' , $C \setminus c_1 \subset C' \neq \emptyset$. Further, because $\nu_\perp$ is (by definition) the least element in C , all elements in C' must be comparable with $\nu_\perp$. Since c_1 includes the hyperbox, and because the regions corresponding to the corners are all unbounded upward closed sets, for some point $\nu' \in C \setminus c_1$, the line between ν_0 and ν must cross a hyperface of the bounding hyperbox. If one of these lines is a witness to the non-convexity, then clearly C is

not comparably convex. Otherwise we construct a new implicant C'' that contains c_1 and C' as follows: for each face that the lines between ν_0 and C' cross, set the polarity of the controlling corner in c_1 to be positive. Since the upward closure from the face was included in C' (and because this only adds elements to c_1) this new implicant must contain both c_1 and C' . However if c_1 and C' are implied by a another implicate they must not have been prime which is a contradiction! ■

Therefore, if $\varphi_B \implies \varphi_C$ and C is comparably convex, then C is a conjunction of the vertices (or their negation). ■

⁹The key property of the prime implicant is that no subset of the conjunctions, c_i , are implied by another (potentially new) conjunction

¹⁰Since such a logical implication is equivalent to a subset relationship