Information and Computation ••• (••••) ••••••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Survey on mining signal temporal logic specifications

Ezio Bartocci [a], Cristinel Mateis [b], Eleonora Nesterini [a,b,*], Dejan Nickovic [b]

[a] Technische Universität Wien, Vienna, Austria
[b] Austrian Institute of Technology, Vienna, Austria

ARTICLE INFO

Article history:
Received 4 December 2021
Received in revised form 4 July 2022
Accepted 4 September 2022
Available online xxxx

ABSTRACT

Formal specifications play an essential role in the life-cycle of modern systems, both at the time of their design and during their operation. Despite their importance, formal specifications are only partially (if at all) available. Specification mining is the process of learning likely system properties from the observation of its behavior and its interaction with the environment. Signal temporal logic (STL) is a popular formalism for expressing properties of cyber-physical systems (CPS). In the last decade, the introduction of first methods for mining STL specifications from time series generated by CPS led to a new vivid area of research.

This survey paper overviews methods for mining STL specifications from CPS behaviors, sketches different approaches found in the literature and presents them in an intuitive and didactic manner. It aims at presenting the most influential techniques and covers most important aspects of specification mining: template-based vs. template-free, model-based vs. model-free, passive vs. active, and supervised vs. unsupervised learning.

1. Introduction

Formal specifications play an important role in any system's life-cycle, including its design, implementation and operation. During the concept design phase, formal specifications can improve the process of engineering requirements, by making them more precise and rigorous. During the system implementation, formal specifications can be used for formal verification of critical components, but also as oracles in the testing activities. Finally, formal specifications can be monitored during system operation to detect violations of requirements and possibly take corrective actions.

Yet, it is often the case that formal specifications are not at all or only partially available. *Specification mining* [1,33,9,38,65,36] is the process of inferring likely system properties from observing its execution and the behavior of its environment. There are benefits of mining specifications both from familiar, well-understood and from unfamiliar systems. Specifications mined from a familiar system can be used (1) to complete the existing incomplete or outdated specifications, (2) for system maintenance, (3) to confirm expected behaviors, (4) to detect bugs, and (5) to generate new tests. Mined properties from unfamiliar systems are typically used for (1) system comprehension, (2) system modeling, (3) reverse engineering, (4) explaining the essential properties of black-box components (e.g., machine learning components).

In this paper, we present the state-of-the-art on mining Signal Temporal Logic (STL) [51] specifications from cyber-physical systems (CPS) behaviors. More specifically, we consider behaviors to be time series representing the continuous real-valued dynamics of the CPS. STL is a popular declarative language for expressing CPS properties by defining temporal and timed relations in time series. In the context of CPS, specification mining has been used for many applications,

* Corresponding author.
E-mail address: eleonora.nesterini@tuwien.ac.at (E. Nesterini).

https://doi.org/10.1016/j.ic.2022.104957

Please cite this article as: E. Bartocci, C. Mateis, E. Nesterini et al., Survey on mining signal temporal logic specifications, Information and Computation, https://doi.org/10.1016/j.ic.2022.104957

Table 1
Mining STL specifications - overview of methods.

Method	Tool	Ref.	Year	STL fragment	Param only	Whole formula	Pos ex only	Also neg ex	Offline	Online	Model	Section
exact val. domain	–	[6]	2011		✓		✓		✓			3.1
approx. val. domain	Breach	[6]	2011	monotonic PSTL	✓		✓		✓			3.2
tightness metrics	TeLEx	[35]	2017		✓		✓		✓			3.3
computation graph	stlcg	[49]	2020	no time parameters	✓		✓		✓			3.4
falsification	S-TaLiRo Breach	[78] [37,38]	2012 2013	monotonic PSTL	✓		✓			✓	✓	3.5
logical clustering	–	[75] [76]	2017 2018	monotonic PSTL	✓		✓		✓			3.6
DAG + supervised learning	–	[46,45]	2014	rPSTL		✓	✓	✓	✓	✓		4.1
DAG + unsupervised learning	–	[39]	2014	iPSTL		✓		✓	✓			4.2
grid-based clustering	–	[74]	2017			✓	✓		✓			4.3
supervised decision trees	–	[19,18,17]	2016	1^{st}, 2^{nd}-level primitive candidates set		✓		✓	✓	✓		4.4.1, 4.4.2
unsupervised decision trees	–	[16]	2017	1^{st}, 2^{nd}-level primitive candidates set		✓	✓		✓			4.4.3
enumerative search	–	[54]	2020	monotonic PSTL		✓	✓		✓			4.5
robustness metric	TeLEx	[36]	2019	predicate candidates set		✓	✓		✓			4.6
genetic algorithm + GP-UCB	–	[59]	2018			✓		✓	✓			4.7
genetic algorithm	–	[68,7]	2019	ptPSTL		✓		✓	✓			4.7

including to automate difficult tasks such as fault-localization [13,38], failure explanation [14], anomaly detection [19,61], verification [12] and control synthesis [22].

This paper is presented in the form of a survey, collecting the relevant methods and tools, providing the reader with the necessary references, but also sketching the main ideas behind the reviewed techniques.

An overview of the specification mining methods

We present 15 procedures for mining STL specifications from 21 scientific papers. The methods presented in this paper are influenced by the specificity of CPS applications, the time series model of the behavior and the choice of STL as the specification language. An overview of all studied methods is given in Table 1. There are multiple orthogonal dimensions of mining STL specifications that we briefly discuss in the remainder of this section. In some cases, when we consider that two or more methods are sufficiently similar, we select and present only one in more detail.

Template-based vs. template-free mining This is the main axe that we use in this paper to classify the specification learning methods. In template-based mining, the user provides the specification skeleton and the mining procedure consists in inferring specification parameters that are consistent with observed data. Template-free mining consists in learning both the structure of the specification as well as its parameters.

Passive vs. active mining Passive mining consists in learning a specification from a fixed set of behaviors. Active mining uses the system (or the system model), assumed to be available, to generate new behaviors and actively steer the inference process.

Supervised vs. unsupervised mining In supervised specification mining, system behaviors are labeled and the aim is to learn specifications that minimize the deviation from the labels. We further distinguish between two main scenarios for supervised mining - using only positive and using both positive and negative examples. Unsupervised learning aims at separating behaviors into multiple clusters according to their similarity, and learning a specification per cluster, where the inferred formula minimizes the mis-classification rate.

Offline vs. online mining Offline mining consists in learning the specification once from the set of available behaviors. The specification remains fixed once inferred. In online mining, it is assumed that new behaviors arrive over time and the mined specification is updated and refined to take into account the effect of new behaviors.

2. Background

In this section, we provide some technical background needed to sketch and describe the specification mining methods. We define signals and systems, introduce *signal temporal logic* (STL), its parametric version PSTL, and its quantitative semantics interpretation.

2.1. Signals and systems

Let $S = \{s_1, \ldots, s_n\}$ be a set of signal variables. We define the time domain $\mathbb{T}$ to be of the form $[0, d] \subset \mathbb{R}$. An n-dimensional *signal* w is a function $\mathbb{T} \to \mathbb{R}^n$, which we also consider to be a vector of real-valued signals $w_i : \mathbb{T} \to \mathbb{R}$ associated to variables s_i for $i = 1, \ldots, n$. Given two signals $w_1 : \mathbb{T} \to \mathbb{R}^l$ and $w_2 : \mathbb{T} \to \mathbb{R}^m$, we define their parallel composition $w_1 \| w_2 : \mathbb{T} \to \mathbb{R}^{l+m}$ in the expected way. Given a signal $w : \mathbb{T} \to \mathbb{R}^n$ and variables $R \subseteq S$ with $R = \{s_{i_1}, \ldots, s_{i_m}\}$ for some $1 \leq i_1 < \ldots < i_m \leq n$, we define the projection of w onto R as $w_R = w_{i_1} \| \ldots \| w_{i_m}$.

A *system* M is an input-output state machine that generates dynamic behaviors – it takes as input a signal u and outputs a signal $w = M(u)$.

2.2. Signal temporal logic

Signal Temporal Logic (STL) [51] is an extension of Linear Temporal Logic (LTL) [66]. STL expands LTL in three directions: (1) it uses time-bounded temporal operators and is interpreted over dense-time, like Metric Temporal Logic (MTL) [47] and Metric Interval Temporal Logic (MITL) [2], (2) it uses numeric predicates that allow to reason about real-valued variables in addition to propositional variables, and (3) it naturally admits quantitative semantics [24]. STL with quantitative semantics enables measuring how far is an observed behavior from satisfying or violating a specification.

Let Θ be a set of terms of the form $f(R)$ where $R \subseteq S$ are subsets of variables and $f : \mathbb{R}^{|R|} \to \mathbb{R}$ are interpreted functions. The syntax of STL is given by the grammar

$$\varphi ::= \mathbf{true} \mid f(R) > k \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{S}_I \varphi_2 \,,$$

where $f(R)$ are terms in Θ, k is either a constant in $\mathbb{Q}$ or a magnitude parameter and I are intervals with bounds that are either constants in $\mathbb{Q} \cup \{\infty\}$ or timing parameters. As customary, the timing interval I may be omitted when $I = [0, \infty)$ or $I = (0, \infty)$.

With respect to a signal w, the semantics of an STL formula is described via the satisfiability relation $(w, i) \models \varphi$, indicating that the signal w satisfies φ at the time t:

$$\begin{aligned}
&(w,t) \models \mathbf{true} \\
&(w,t) \models f(R) > 0 && \text{iff} && f(w_R[t]) > 0 \\
&(w,t) \models \neg\varphi && \text{iff} && (w,t) \not\models \varphi \\
&(w,t) \models \varphi_1 \vee \varphi_2 && \text{iff} && (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
&(w,t) \models \varphi_1 \mathbf{U}_I \varphi_2 && \text{iff} && \exists t' \in t \oplus I,\ (w,t') \models \varphi_2 \text{ and} \\
& && && \forall t'' \in (t,t'),\ (w,t'') \models \varphi_1 \\
&(w,t) \models \varphi_1 \mathbf{S}_I \varphi_2 && \text{iff} && \exists t' \in t \ominus I,\ (w,t') \models \varphi_2 \text{ and} \\
& && && \forall t'' \in (t',t),\ (w,t'') \models \varphi_1 \,.
\end{aligned}$$

Here we use the symbol $\oplus$ to denote the Minkowski sum extended to be defined between the scalar t and the set I as follows: $t \oplus I = \{t + a \mid a \in I\}$. Analogously, the symbol $\ominus$ denotes the Minkowski difference: $t \ominus I = \{t - a \mid a \in I\}$. We write $w \models \varphi$ when $(w, 0) \models \varphi$.

We use **S** and **U** as syntactic sugar for the *untimed* variants of the *since* $\mathbf{S}_{(0,\infty)}$ and *until* $\mathbf{U}_{(0,\infty)}$ operators. From the basic definition of STL, we can derive the following standard operators.

tautology	**true**	=	$p \vee \neg p$
contradiction	**false**	=	$\neg$**true**
conjunction	$\varphi_1 \wedge \varphi_2$	=	$\neg(\neg\varphi_1 \vee \neg\varphi_2)$
implication	$\varphi_1 \rightarrow \varphi_2$	=	$\neg\varphi_1 \vee \varphi_2$
eventually, finally	$\mathbf{F}_I\varphi$	=	**true** $\mathbf{U}_I\, \varphi$
always, globally	$\mathbf{G}_I\varphi$	=	$\neg\mathbf{F}_I\neg\varphi$
once	$\mathbf{O}_I\varphi$	=	**true** $\mathbf{S}_I\, \varphi$
historically	$\mathbf{H}_I\varphi$	=	$\neg\mathbf{O}_I\neg\varphi$
rising edge	**rise**(φ)	=	$\varphi \wedge \neg\varphi\, \mathbf{S}$ **true**
falling edge	**fall**(φ)	=	$\neg\varphi \wedge \varphi\, \mathbf{S}$ **true**

2.3. Parametric signal temporal logic and its validity domains

In this section, we show how to extend STL to its *parametric* variant PSTL, following the approach introduced in [6]. Let $P = \{p_1, \ldots, p_m\}$ be a set of magnitude parameters and $Q = \{q_1, \ldots, q_k\}$ a set of timing parameters, defined over their respective domains $\mathcal{P}$ and $\mathcal{Q}$, say hyper-rectangles in $\mathbb{R}^m$ and $\mathbb{R}^k$. PSTL extends STL by allowing the amplitude constant k in $f(R) > k$ to be replaced by a magnitude parameter p and the bound constants in $\mathbf{U}_I$ and $\mathbf{S}_I$ to be replaced by timing parameters in Q. A parameter valuation $(\mathbf{u}, \mathbf{v}) \in \mathcal{P} \times \mathcal{Q}$ transforms a PSTL formula φ to an STL formula $\varphi_{\mathbf{u},\mathbf{v}}$ by substituting the values $(\mathbf{u}, \mathbf{v})$ in the parameters $(\mathbf{p}, \mathbf{q})$.

The semantics of PSTL formulas can be now defined using the notion of a *validity domain* [26] $\mathcal{D}(\varphi, w) \subseteq \mathcal{P} \times \mathcal{Q}$ that consists of all parameter valuations $(\mathbf{u}, \mathbf{v}) \in \mathcal{P} \times \mathcal{Q}$ such that w satisfies the STL formula $\varphi_{\mathbf{u},\mathbf{v}}$. In order to compute validity domains, we introduce the extended validity domain $D(\varphi, w) \subseteq \mathcal{P} \times \mathcal{Q} \times \mathbb{T}$, where $(\mathbf{u}, \mathbf{v}, t) \in D(\varphi, w)$ iff $(w, t) \models \varphi_{\mathbf{u},\mathbf{v}}$. The extended validity domain $D(\varphi, w)$ is defined inductively as follows:

$$
\begin{aligned}
D(\mathbf{true}, w, t) &= \mathcal{P} \times \mathcal{Q} \times \mathbb{T} \\
D(f(R) \geq k, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid f(R(t)) \geq k_{\mathbf{u},\mathbf{v}}\} \\
D(\neg\varphi, w, t) &= \bar{D}(\varphi, w, t) \\
D(\varphi_1 \vee \varphi_2, w, t) &= D(\varphi_1, w, t) \cup D(\varphi_2, w, t) \\
D(\varphi_1 \mathbf{U}_I \varphi_2, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid \exists t' \in t \oplus I_{\mathbf{u},\mathbf{v}} \text{ s.t. } (\mathbf{u}, \mathbf{v}, t') \in D(\varphi_2, w, t') \wedge \\
&\qquad \forall t'' \in [t, t'] \, (\mathbf{u}, \mathbf{v}, t'') \in D(\varphi_1, w, t'')\} \\
D(\varphi_1 \mathbf{S}_I \varphi_2, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid \exists t' \in t \ominus I_{\mathbf{u},\mathbf{v}} \text{ s.t. } (\mathbf{u}, \mathbf{v}, t') \in D(\varphi_2, w, t') \wedge \\
&\qquad \forall t'' \in [t', t] \, (\mathbf{u}, \mathbf{v}, t'') \in D(\varphi_1, w, t'')\}
\end{aligned}
$$

2.4. Signal temporal logic and its quantitative semantics

Apart from its classical qualitative semantics, presented in Section 2.2, STL also admits *quantitative* semantics [27,24]. The STL quantitative semantics is expressed in terms of a *robustness function* $\rho(\varphi, w, t)$ that assumes values in the set $\mathbb{R} \cup \{+\infty, -\infty\}$ of extended reals and measures how far is the signal w from violating or satisfying the specification φ at time t. It is defined inductively as follows:

$$
\begin{aligned}
\rho(\mathbf{true}, w, t) &= +\infty \\
\rho(f(R) > 0, w, t) &= f(w_R[t]) \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi_1 \vee \varphi_2, w, t) &= \max(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\
\rho(\varphi_1 \mathbf{U}_I \varphi_2, w, t) &= \sup_{t' \in t \oplus I} \min(\rho(\varphi_2, w, t'), \inf_{t'' \in (t, t')} \rho(\varphi_1, w, t'')) \\
\rho(\varphi_1 \mathbf{S}_I \varphi_2, w, t) &= \sup_{t' \in t \ominus I} \min(\rho(\varphi_2, w, t'), \inf_{t'' \in (t', t)} \rho(\varphi_1, w, t''))
\end{aligned}
$$

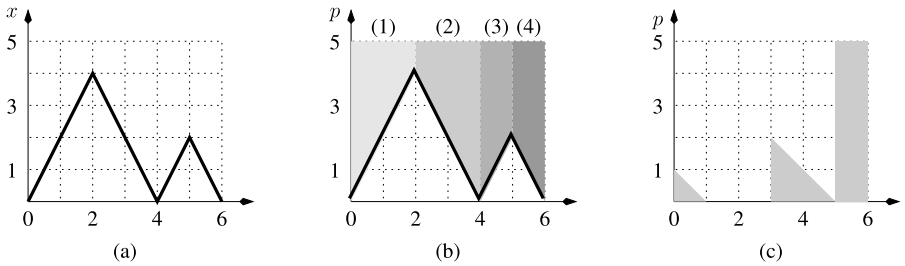

Fig. 1. Exact computation of validity domains [6]: (a) example trace w, (b) validity domain $\mathcal{D}(x > p, w)$, and (c) validity domain $\mathcal{D}(\mathbf{G}_{[0,p]} x > 2, w)$.

The quantitative semantics of STL have two fundamental properties. The first property states that the STL quantitative semantics definition is *sound* in the sense of [28] – the sign of the robustness degree $\rho(\varphi, w, t)$ (whenever it is different from zero) indicates the satisfaction status of the specification. Second, whenever w satisfies φ at time t with some (positive) robustness degree k, any other trace w' whose infinity-norm distance from w is smaller than k also satisfies φ at time t, where the infinity-norm distance between signals w and w' is defined as follows: $\|w - w'\|_\infty = \sup_{t \in \mathbb{T}} |w(t) - w'(t)|$.

3. Mining parameters in PSTL specification templates

Template-based inference of STL specifications represents the first big family of specification mining methods. In this setting, the PSTL formula template is given by the user, and the mining procedure learns the parameter values that instantiate the template to a concrete STL specification.

3.1. Exact computation of validity domains

The first mining procedure computes the exact validity domain of a PSTL specification φ given a signal w [6]. We consider the restricted class of *sampled* signals given as a finite sequence of (time stamp, value) pairs $(t_1, w(t_1)), \ldots, (t_l, w(t_l))$ for an increasing sequence of time stamps with $t_1 = 0$ and with the values between two consecutive time stamps being defined using *linear interpolation*. It is shown that under this restriction, the validity domains are semi-linear, meaning that they can be expressed as Boolean combinations of linear constraints over corresponding variables.

The computation of the exact validity domain follows closely the inductive definition of the extended validity domain and consists of three main steps:

1. *Base case:* we first compute the validity domain of the numeric predicates $f(R) \geq k$, using the transformation
$$\bigvee_{i=1}^{l-1} t_i \leq t < t_{i+1} \wedge \alpha_i t + \beta_i \geq k,$$
where $\alpha_i = \frac{w[t_{i+1}] - w[t_i]}{t_{i+1} - t_i}$ and $\beta_i = \frac{t_{i+1} w[t_i] - t_i w[t_{i+1}]}{t_{i+1} - t_i}$.
2. *Inductive step:* we apply the semantic definitions of extended validity domain for Boolean and temporal operators. This results in a quantified linear real arithmetic formula.
3. *Quantifier elimination:* we eliminate the quantifiers from the formula using an off-the-shelf satisfiability modulo theory (SMT) solver, obtaining a quantifier-free semi-linear constraint.

The resulting semi-linear constraint characterizes exactly the validity domain $\mathcal{D}(\varphi, w)$. Fig. 1 illustrates the outcomes of the exact validity domain computation. Given the signal w, characterized by the sequence of sampling points $(0, 0), (2, 4), (4, 0), (5, 2), (6, 0)$ (Fig. 1 (a)), the validity domain $\mathcal{D}(x > p, w)$ is characterized by the semi-linear constraint (depicted in Fig. 1 (b)):

$$\begin{array}{lll} (0 \leq t < 2 \wedge p \leq 2t) & \vee & (1) \\ (2 \leq t < 4 \wedge p + 2t \leq 8) & \vee & (2) \\ (4 \leq t < 5 \wedge p + 8 \leq 2t) & \vee & (3) \\ (5 \leq t < 6 \wedge p + 2t \leq 12) & & (4) \end{array}$$

The validity domain $\mathcal{D}(\mathbf{G}_{[0,p]} x > 2, w)$ is shown in Fig. 1 (c) and is given by the semi-linear constraint:

$$\begin{array}{l} (0 \leq t < 1 \wedge p < 1 - t) \ \vee \\ (3 \leq t < 5 \wedge p < 5 - t) \ \vee \\ (5 \leq t < 6 \wedge 0 \leq p < 5) \end{array}$$

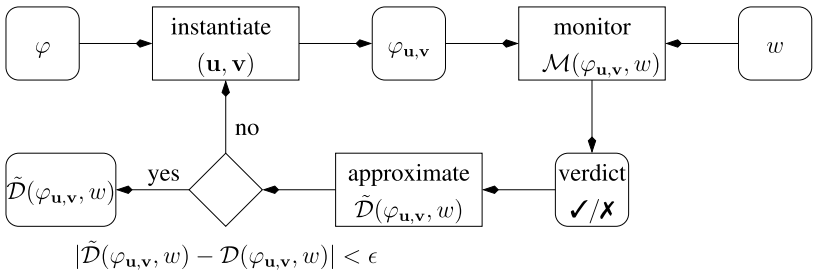

Fig. 2. Workflow for approximating validity domain using monitoring [6].

Table 2
PSTL formulas and their polarity.

Polarity		Formula
positive	:	$x < p$
		$\mathbf{F}_{[0,p]}(x < 2)$
negative	:	$x > p$
		$\mathbf{G}_{[0,p]}(x < 2)$
undetermined	:	$(f(x) > p) \wedge (g(x) < p)$

This concludes the description of the basic problem that consists in passively learning parameters from positive examples. The price paid for obtaining the exact validity domain is a high computational cost. In fact, eliminating quantifiers from linear real arithmetic formulas is exponential in the size of the formula and the number of alternating quantifiers [29,50].

3.2. Approximating validity domains with monitoring and search

The high cost of computing exactly validity domains can be mitigated by approximating them using runtime verification methods combined with search [6]. We assume a monitoring procedure $\mathcal{M}(\varphi, w)$ that takes as inputs an STL specification φ and a signal w. The monitor outputs **true** if $w \models \varphi$ and **false** otherwise. There are many publicly available implementations of STL monitoring libraries, including RTAMT [63], Reelay [73], MoonLight [8,58], Breach [23], S-TaLiRo [4].

The monitoring procedure can be combined with the parameter sampling to explore the parameter space of the specification and approximate the validity domain of the PSTL specification relative to a specific signal. Fig. 2 depicts the overall idea of the iterative approach in which each iteration consists of the following steps:

1. selection of a parameter $(\mathbf{u}, \mathbf{v})$ and instantiation of the PSTL formula φ to obtain the STL formula $\varphi_{\mathbf{u},\mathbf{v}}$,
2. application of the monitoring procedure $\mathcal{M}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ to check whether the trace w satisfies the instantiated specification, and
3. approximation $\tilde{\mathcal{D}}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ of the exact validity domain depending on the monitoring verdict.

The algorithm stops when the approximated validity domain $\tilde{\mathcal{D}}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ is ϵ-close to the exact validity domain $\mathcal{D}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ for some user-defined threshold ϵ. While the exact validity domain is not known and hence cannot be directly used in the computation of the ϵ-closeness, its area can be bounded in each iteration and used to conservatively estimate the measure. This high-level description of the procedure does not explain how to perform its two main steps: (i) the approximation of the validity domain from the monitoring verdict, and (ii) the estimation of the difference between the approximated and the exact validity domain. We discuss next these two steps.

To efficiently approximate a validity domain using the monitoring-based parameter exploration, we need to consider a well-behaving subset of PSTL, that we call *monotonic* PSTL, following the naming convention introduced in [75]. The definition of the monotonic PSTL fragment is based on the notion of polarity [6] of a PSTL formula φ: the *polarity* $\pi(p, \varphi)$ of a parameter p with respect to a PSTL formula φ is *positive* (*negative*) if it is easier to satisfy the formula as we increase (decrease) the value of p. In other words, for all φ with positive (negative) polarity, and all $p \geq p'$, we have that $\varphi_{p'} \rightarrow \varphi_p$ $(\varphi_p \rightarrow \varphi_{p'})$.

A PSTL formula φ is said to have *undetermined* polarity $\pi(p, \varphi)$ of a parameter p with respect to a PSTL formula φ if neither increasing nor decreasing the value of the parameter makes the satisfaction of the formula easier. Table 2 shows several examples of PSTL formulas and their polarity. For example, $x < p$ has positive polarity. It follows that $x < 1$ implies $x < 10$.

We say that a PSTL specification φ is monotonic in a parameter p if $\pi(p, \varphi)$ is determined. Similarly, a PSTL specification φ is said to be monotonic if for every parameter p, $\pi(p, \varphi)$ is determined. A monotonic PSTL specification φ induces a monotonic validity domain. This means that if the parameter instantiation $(v_1, \ldots, v_i, \ldots, v_n) \in \mathcal{P} \times \mathcal{Q}$ is in the validity domain $\mathcal{D}(\varphi, w)$ and the parameter p_i has positive (negative) polarity then $(v_1, \ldots, v_i', \ldots, v_n)$ is also in $\mathcal{D}(\varphi, w)$ for all

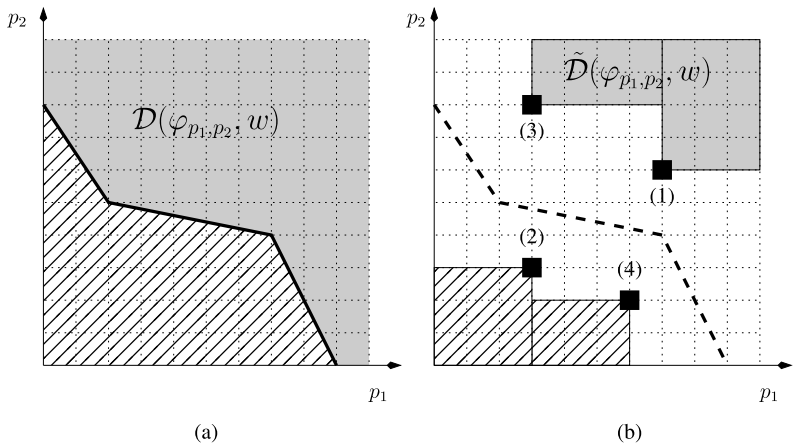

Fig. 3. An example of validity domain approximation using monitoring.

$v_i' > v_i$ ($v_i' < v_i$). In practice, this means that a single check whether a parameter instantiation is in the validity domain allows to infer the membership of an entire hyper-box of parameter values to that validity domain. We illustrate this idea on an example as follows.

Fig. 3 (a) shows an exact validity domain (gray area) that we want to approximate. Fig. 3 (b) depicts four steps of the approximation procedure. In each step, a sample (v_1, v_2) in the parameter space (p_1, p_2) is selected. We instantiate the PSTL formula φ to φ_{v_1,v_2}. We then use the monitoring procedure $\mathcal{M}(\varphi_{v_1,v_2}, w)$ to check whether the signal w satisfies the instantiated specification φ_{v_1,v_2}, i.e. whether $(v_1, v_2) \in \mathcal{D}(\varphi, w)$. After each membership check, we can infer regions of parameter values that are inside or outside the validity domain. These regions correspond to the four rectangles associated to the parameter samples, namely gray rectangles for the samples (1) and (3) and dashed rectangles for the samples (2) and (4). The white region is the uncertainty region. The area of the uncertainty region provides a distance to the exact validity domain. That is, the smaller the uncertainty region, the closer we are to the exact validity domain.

The specification mining method using the approximate computation of the validity domain presented in this section is implemented in the tool Breach [23].

3.3. Inferring PSTL parameters using tightness metrics

Passive learning of PSTL parameters from positive examples has the risk of over-generalization. Selecting a parameter valuation from the validity domain is not a trivial task. The methods for identifying PSTL parameters presented in Sections 3.1 and 3.2 compute (or approximate) the entire validity domain and the selection task is left to the engineer. Good candidates for parameter values that avoid over-generalization are the points lying on the boundary of the validity domain. However, there are many of them and they are partially ordered.

In this section, we present the work that addresses the problem of passively inferring the optimal parameter values for a PSTL specification from a set of positive examples [35], without computing the entire validity domain. To this goal, the authors introduce the notion of *tightness* – ϵ-tight satisfaction of an STL specification φ by a signal w requires the existence of a magnitude or timing bound in φ whose perturbation by ϵ results in the violation of φ by w. In this setting, the parameter mining problem can be formulated as follows: given a PSTL specification template φ and a signal w, find a parameter valuation $\mathbf{v}$ such that w ϵ-tightly satisfies the STL formula $\varphi_{\mathbf{v}}$. The solution to this problem requires to solve an optimization problem where the objective function is the absolute value of the robustness degree of the mined formula under the additional constraint that the formula is satisfied. However, this optimization problem is hard to solve because the objective function is non-differentiable.

To remedy this problem, the authors embed a *tightness measure* as part of the new quantitative semantics for STL. The main idea is to make predicates and temporal operators *smooth*, hence differentiable. This enables the use of more efficient gradient-based numerical optimization algorithms (e.g. quasi-Newton) to solve the problem of inferring the STL specification with ϵ-tight satisfaction. The resulting tightness metric is shown to be sound, i.e. the adapted robustness degree is non-negative iff the formula is satisfied. This approach is implemented in the tool TeLEx [35].

3.4. Mining specification parameters with computation graphs

This section presents the framework stlcg [49] introduced to efficiently evaluate the robustness of STL formulas using the computation graphs from machine learning domain. The built-in backpropagation mechanism of the computation graphs allows for learning the parameter values of a given PSTL formula which e.g. maximize the STL robustness w.r.t. a given set of signal examples. In general, it is possible to infer parameter values w.r.t. any, possibly non-linear, differentiable constraints

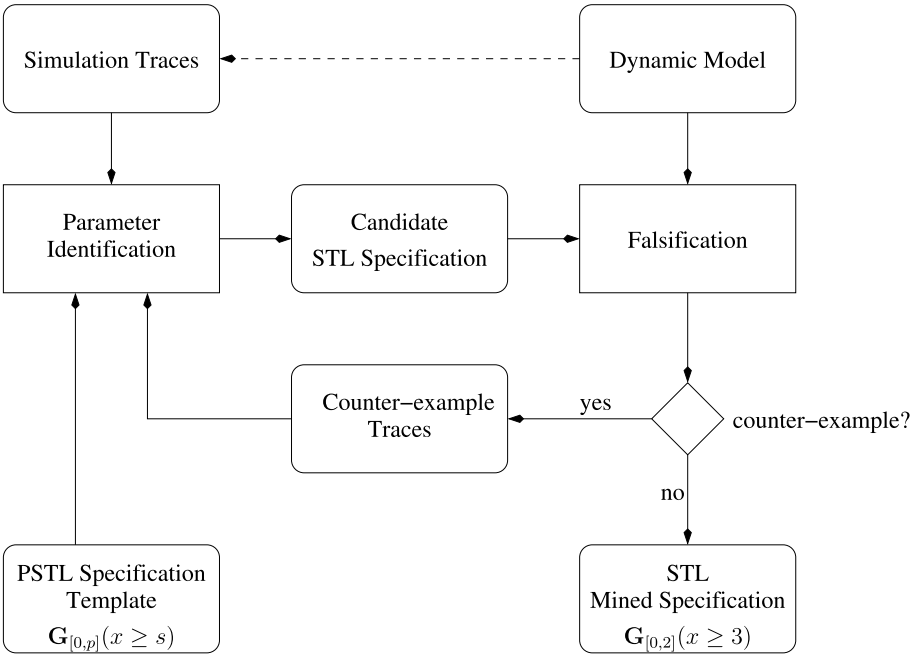

Fig. 4. Overview of specification mining from a control model [37,38].

which can be encoded into the objective function of an optimization problem. We note that stlcg also enables a seamless integration of the PSTL framework into any machine learning problem which might use STL to achieve a particular objective, e.g. using the STL robustness as a reward function in a reinforcement learning problem. stlcg currently does not support the mining of time parameters which is mentioned to be covered in future work.

3.5. Mining specification parameters from dynamic models

So far, all presented methods assumed a passive learning approach with a fixed set of examples from which a STL specification is mined. In this section, we present an active learning approach to mining PSTL parameters from signals. The approach assumes the availability of a dynamic model that can be simulated and that can generate new signals on demand. This is in contrast to the previously described approaches that assume a fixed set of signals. The generation of new signals from the model enables refining the learned parameters.

The specification mining problem is stated as follows: given a dynamic model M and a PSTL specification template φ, find an STL instantiation $\varphi_{\mathbf{u},\mathbf{v}}$ of φ such that *all traces* generated by M satisfy $\varphi_{\mathbf{u},\mathbf{v}}$. This problem was originally stated and solved [78] with the restriction to the specification templates with a single parameter. The approach was subsequently generalized to an arbitrary number of parameters, providing an efficient solution in the case of monotonic PSTL [37,38].

The problem is solved using an iterative procedure, illustrated in Fig. 4. The procedure takes as input a PSTL specification template and a dynamic model. In every iteration, it performs the following steps:

- *Step 1:* compute a candidate STL specification, using for example the mining procedure presented in Section 3.2.
- *Step 2:* check whether the dynamic model can generate a simulation that violates the candidate specification, using *falsification* [60].
- *Step 3:* if the falsification method does not find a counter-example to the candidate specification, accept and return that specification. Otherwise, add the counter-example to the set of simulation traces and repeat from Step 1.

The key step in this approach is the application of the falsification method. The falsification problem is formulated as follows: given an STL φ, find an input signal u such that $M(u)$ violates φ. To solve efficiently this problem, the STL specification is interpreted using quantitative semantics, and the search for the input u is formulated as the following optimization task:

$$\rho^* = \min_u \rho(\varphi, M(u)).$$

If the resulting $\rho^*(\varphi, M(u)) < 0$, we return the input $u^* = \text{argmin}_u \rho(\varphi, M(u))$ that yields a counter-example simulation trace. The different flavors of the specification mining method presented in this section are implemented in the tools S-TaLiRo [4] and Breach [23].

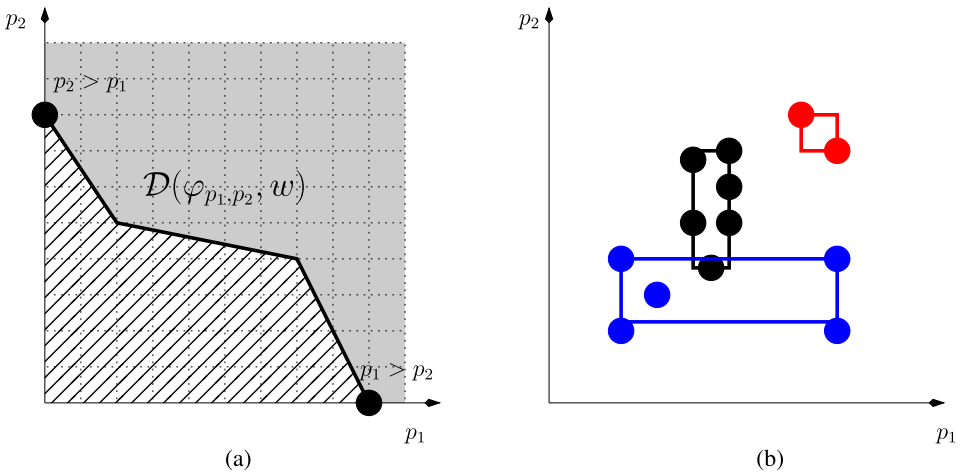

Fig. 5. Example of logical clustering: (a) representative template parameters for different lexicographic orders, and (b) hyper-rectangle approximation of clusters.

3.6. Logical clustering with PSTL

Logical clustering is a method proposed in [75] that combines parameter inference in PSTL specification templates with unsupervised machine learning to group and characterize similar qualitative signals. The logical clustering procedure takes as input a monotonic PSTL specification template φ and a set of signals $\mathcal{W}$. The core idea of the procedure is very simple and consists of the following steps:

- *Step 1:* for each signal $w \in \mathcal{W}$, project the signal to a representative template parameter $\mathbf{v}$ in the validity domain $\mathcal{D}(\varphi, w)$.
- *Step 2:* apply an off-the-shelf clustering method with representative template parameters as features to group together qualitatively similar signals under the lens of the PSTL specification.
- *Step 3:* for each cluster, define an STL instantiation of φ that characterizes well the cluster.

There are two difficulties in realizing the above algorithm sketch: (i) finding the single point in the validity domain that is a good qualitative representative of the signal behavior, and (ii) finding an appropriate STL characterization of a cluster.

The intuition behind Step 1 is to project the signal w to a parameter valuation $\mathbf{v}$ that yields an STL instantiation $\varphi_{\mathbf{v}}$ of the PSTL template φ such that w only marginally satisfies $\varphi_{\mathbf{v}}$ under the quantitative semantics interpretation. The choice of using the monotonic fragment of PSTL plays a key role in this step – a parameter p with determined polarity in φ induces a total order in its domain. Given a signal w and a PSTL specification φ with a single determined parameter p, the extreme value of p in $\mathcal{D}(\varphi, w)$ is arguably the best representative template parameter. The problem is that many PSTL specification templates have more than one parameter. Parameters are often independent from each other and valuations of different parameters induce a partial order. This issue can be addressed by linearizing a partial order into a total lexicographic order. Fig. 5 (a) shows a 2-dimensional validity domain $\mathcal{D}(\varphi, w)$ for some PSTL formula and signal w and two projections of w to representative template parameters according to two lexicographic orders $p_1 > p_2$ and $p_2 > p_1$.

The off-the-shelf clustering methods do not yield an outcome that can be easily plugged in interpretable STL characterizations of the resulting clusters. For a given set of points in the cluster, the tightest-enclosing hyper-rectangle, as shown in Fig. 5 (b), is a useful approximation of the cluster. The hyper-rectangle description can be embedded in the STL formula in a straight-forward manner by using intervals, thus providing a specification that is interpretable by the human. We note that the hyper-rectangles from different clusters can overlap (see Fig. 5 (b)). This can be addressed either by defining a new cluster for the points in the intersection, or by requiring additional guidance from the designer.

The logical clustering method is extended in [76] by dropping the lexicographic ordering in favor of a more general ordering-free approach. Given a PSTL specification template φ and two signals w and w', the authors introduce the notion of a *logical distance* $d_\varphi(w, w')$, which is the Hausdorff distance between the boundaries of the validity domains $\mathcal{D}(\varphi, w)$ and $\mathcal{D}(\varphi, w')$ and can be approximated with arbitrary decision.

4. Mining complete STL specifications

In this section, we present the second major family of specification mining methods that infer the entire STL formula for observed signals, without following a fixed template. All template-free mining methods follow a similar approach. First, the structure of a PSTL specification template is learned from data. Second, the PSTL template is instantiated to a concrete STL specification, using parameter inference methods similar to the ones described in Section 3. Most of these approaches

consider only specific fragments of STL that enable the use of specific optimization techniques. Mining STL specifications with the *until* operator results to be quite challenging and it requires the use of special metaheuristics such as genetic algorithms.

4.1. Mining STL specifications for signal classification

We first present a passive learning approach that infers an STL specification from both positive and negative examples [46]. The resulting STL formula serves as a classifier that is able to distinguish between two types of signals.

This work restricts its attention to the *reactive* fragment of STL, which aims to capture the *cause-effect* relations from signals. Reactive STL (rSTL) is defined using the following syntax:

$$\varphi ::= \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e), \tag{1}$$

where φ_c and φ_e are the *cause* and the *effect* formulas of the following form:

$$\varphi_i ::= \mathbf{F}_{[\tau_1,\tau_2)} p \mid \mathbf{G}_{[\tau_1,\tau_2)} p \mid \varphi_i \vee \varphi_i \mid \varphi_i \wedge \varphi_i, \tag{2}$$

where $i \in \{c, e\}$ and p is a linear predicate, i.e. an inequality over a scalar variable. Reactive PSTL (rPSTL) is defined in a similar manner. We observe that both rSTL and rPSTL admit natural partial ordering of specifications with respect to the satisfaction relation. For instance, any signal w that satisfies $\varphi_1 = \mathbf{G}_{[0,a)} x \leq k$, also satisfies $\varphi_2 = \mathbf{F}_{[0,a)} x \leq k$. We denote this fact by $\varphi_1 \preceq \varphi_2$.

The specification mining problem is stated as follows: Given a set of *labeled* signals, infer a rSTL formula $\varphi = \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e)$ that (i) is consistent with the signal labels, and (ii) can be used to predict the label from a prefix of the signal. The procedure assumes that the effect of the rSTL specification is captured in the *suffix* of the signal. The duration d of that suffix is provided by the user. The procedure consists of two separate phases reflecting, respectively, goals (i) and (ii):

- *Step 1:* infer the effect formula φ_e from the suffix of the signals, and
- *Step 2:* given the (now fixed) effect formula φ_e mined in Step 1, infer the full specification $\varphi = \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e)$ that maximizes how well φ predicts the label of training signals.

Both Step 1 and 2 are modeled as optimization problems that maximize the prediction of the label, while minimizing the size of the mined formula (the optimization problem in Step 1 is defined considering only the suffix of the signals, while the one in Step 2 deals with the entire signals). To efficiently solve the two optimization problems, the procedure uses the partial order $\preceq$ of the candidate specifications, represented by a directed acyclic graph (DAG). The DAG contains all candidate formulas as its nodes, while the directed edges represent the partial order relation between formulas. In essence, finding the optimal specification consists in simultaneously searching for the optimal formula structure in the DAG and its optimal parameters. The mined specification should ideally be the lowest order formula that differentiates between positive and negative examples.

The DAG is infinite, hence the procedure constructs a finite sub-graph of possible candidate specifications and expands it on-the-fly when needed. The procedure starts the search from the DAG bottom elements – the most exclusive formulas of the form $\mathbf{G}_{[t_1,t_2]}(x \circ s)$ with $\circ \in \{\leq, \geq\}$. It then follows the directed edges in the graph until a specification satisfied by all the positive examples is found. At each iteration, a candidate template rPSTL formula is explored and its parameters are estimated using classical optimization methods (e.g., simulated annealing). Fig. 6 illustrates the DAG refinement iteration on a simple example. If the resulting rSTL formula correctly classifies all positive examples and only a fraction of the negative examples above some high threshold, the algorithm stops and the candidate specification is selected. Otherwise, the high-cost node is pruned from the DAG and the DAG is expanded to new candidate formulas. The expansion is done by combining nodes that create a specification with higher order with respect to their parent(s) node(s). The procedure is then repeated. This procedure is extended to an online learning setting in [45].

4.2. Mining STL specifications for anomaly detection

We summarize in this section a procedure for unsupervised passive learning of STL specifications [39]. This procedure adapts the supervised learning approach described in Section 4.1 to the setting where signals are not labeled. The aim of this procedure is to learn a specification that characterizes *nominal* behavior of a system. Any signal that violates the mined specification is then considered to be *anomalous*.

The specification mining procedure related to the *inference* fragment of STL is introduced in [39]. Inference STL (iSTL) removes from the rSTL fragment the restriction that top-level formulas must contain a Boolean implication, and is thus defined as follows:

$$\varphi ::= \mathbf{F}_{[0,T)} \varphi$$

where φ has the form of cause or effect formulas (2) in Section 4.1.

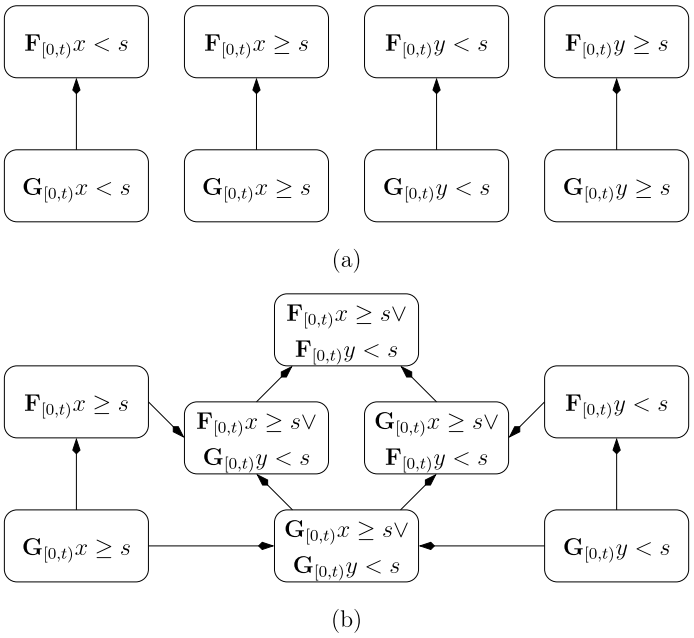

Fig. 6. Example of a finite sub-DAG with candidate specifications built from observing a two-dimensional (x, y)-signal: (a) at initialization, and (b) after a refinement [46].

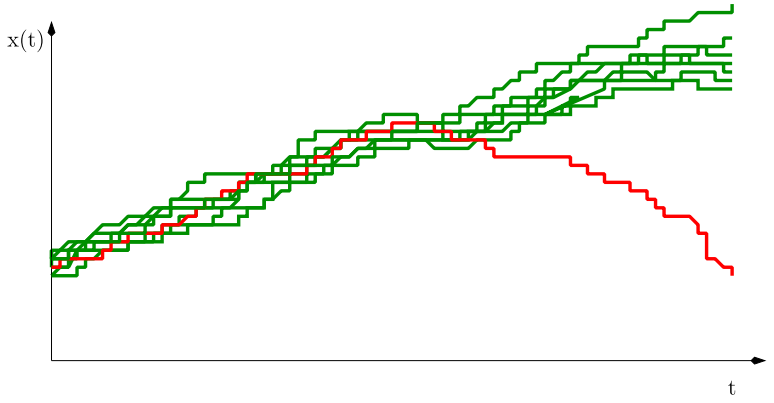

Fig. 7. Example of nominal (green) and anomalous (red) signals. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The specification mining procedure inherits several ideas from the algorithm described in Section 4.1, including the formulation of the mining as an optimization problem and the search of candidate formulas in the DAG representing the partial order of specifications. However, there are two key differences from the previous work.

First, since the signals are not labeled, the procedure makes two assumptions to be able to distinguish between nominal and anomalous behavior: (i) anomalous signals are rare and with low probability, and (ii) the signals associated to nominal behaviors are qualitatively different from the anomalous ones. Fig. 7 depicts these assumptions. Second, the objective function is formalized using one-class support vector machines (SVM). This method moves the optimization problem to a higher-dimensional space that increases the separation between nominal and anomalous behaviors. The objective function is designed to simultaneously

- minimize the number of signals classified as anomalous – this is consistent with the first assumption,
- maximize the separation between nominal and anomalous behavior – this is consistent with the second assumption, and
- penalize, using a tightness metric, the number of signals that satisfy with large robustness the specification – to avoid trivial specifications such as **true** that is satisfied by all signals.

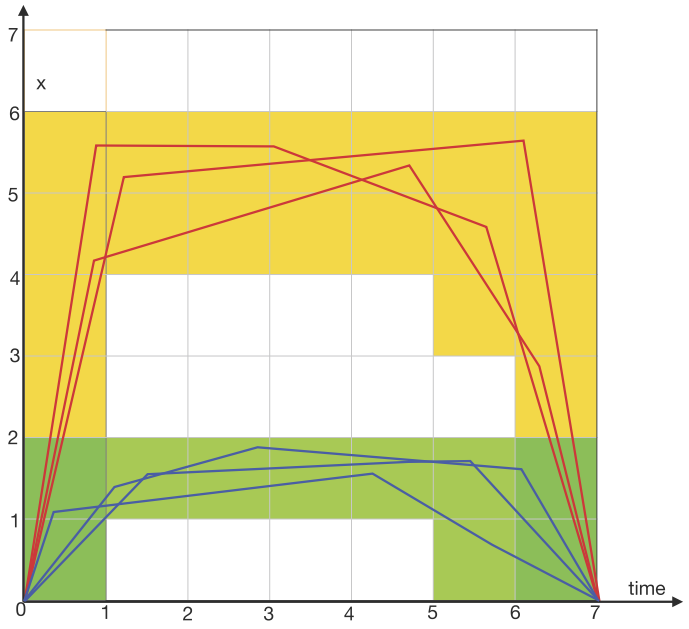

Fig. 8. The STL formula which describes the yellow cluster and therefore the included red signals is: $\varphi_1 = \mathbf{G}_{[0,7]}(x \leq 6) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 4) \wedge \mathbf{G}_{[5,6]}(x > 3) \wedge \mathbf{G}_{[6,7]}(x > 0)$. The one for the green cluster (blue signals) is $\varphi_2 = \mathbf{G}_{[0,7]}(x \leq 2) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 1) \wedge \mathbf{G}_{[5,7]}(x > 0)$. The final formula is obtained by disjunction between the two STL formulas: $\varphi = \varphi_1 \vee \varphi_2$. This example is taken from [74].

4.3. Mining STL specifications using grids

In this section, we present another passive learning approach for mining STL specifications using a *grid-based* discretization of the time and value domain [74].

We sketch the mining procedure, also illustrated in Fig. 8, for the special case of one-dimensional signals:

- *Step 1:* discretize the two-dimensional time-value space into a rectangular grid, with granularity regulated by the user.
- *Step 2:* for every signal, mark all the cells in the grid that it covers (intersects).
- *Step 3:* cluster the signals according to the similarity of cells that they cover, where the exact notion of similarity is defined by the user.
- *Step 4:* for each cluster, map all the cells covered by the signals in that cluster.
- *Step 5:* translate the set of cells associated to each cluster to an STL formula that characterizes the cluster.
- *Step 6:* build the overall STL formula as disjunction of the cluster STL formulas.

The main consideration to take when using this procedure is the granularity of the grid, which affects the tightness of the resulting STL formula. Small granularity may result in a large over-fitting STL formula, while large granularity may result in an STL formula that accepts too many signals and provides a poor characterization of the signals.

Translating the set of cells to an STL formula works as follows. For every column i in the grid (time steps), we identify the bottom-most j and the top-most k rows (value steps) such that the cells (i, j) and (i, k) are in the set. We then describe this column using the STL formula $\mathbf{G}_{[t_1,t_2]}(x \geq s_1 \wedge x \leq s_2)$, where t_1 and t_2 are the start and the end times of column i, s_1 is the minimum value of the row j and s_2 is the maximum value of the row k. The overall STL specification is the conjunction of formulas obtained for each column.

We finally note that the generalization of the mining procedure from one-dimensional to multi-dimensional signals is straight-forward. The presented procedure is applied to each component of the multi-dimensional signal and the resulting formulas from each component are combined using conjunction.

4.4. Mining STL specifications with decision trees

Decision trees present an alternative for mining STL specifications. We review in this section three methods based on decision trees for inferring STL formulas: (i) offline supervised learning from positive and negative examples, (ii) online supervised learning, and (iii) offline unsupervised learning.

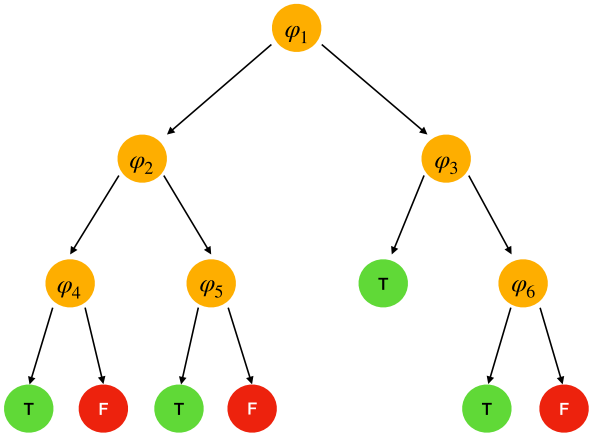

Fig. 9. Example of transformation from binary tree to STL formula as presented in [19]. The inferred formula in this case is: $\varphi_{\text{tree}} = \big(\varphi_1 \wedge ((\varphi_2 \wedge \varphi_4) \vee (\neg\varphi_2 \wedge \varphi_5))\big) \vee \big(\neg\varphi_1 \wedge (\varphi_3 \vee (\neg\varphi_3 \wedge \varphi_6))\big)$. The STL formulas φ_i (for $i \in \{1, \ldots, 6\}$) represent the instantiations of PSTL primitives associated to the corresponding nodes.

4.4.1. *Offline supervised learning*

In this section, we present an approach for passive learning of STL specifications from positive and negative examples using *binary decision trees* [19].

The objective is the same as in Section 4.1 and consists in finding an STL specification that minimizes the mis-classification rate. The specification mining procedure makes use of a key insight – the possibility to map a fragment of STL to decision trees, and more specifically to *binary* decision trees. This enables using classical methods on decision trees to solve the problem – first, build a tree that classifies signals, and then translate the tree to an STL formula.

The procedure takes three parameters as input: (i) a set of PSTL template specifications that are allowed to be used for generating the final specification, (ii) an *impurity* measure that defines the criteria for preferring one PSTL template over another, and (iii) a stopping criterion. Two classes of primitive PSTL formulas are proposed in the original paper [19] – first-level primitives of the form $\mathbf{F}_I(x \circ s)$ or $\mathbf{G}_I(x \circ s)$, and second-level primitives of the form $\mathbf{F}_I\mathbf{G}_J(x \circ s)$ or $\mathbf{G}_I\mathbf{F}_J(x \circ s)$, where $\circ \in \{\leq, \geq\}$. The authors propose canonical measures, including information, Gini and mis-classification gain, as possible impurity measures. These measures are adapted to better generalize unseen data by defining boundaries that separate two classes of signals as much as possible based on the robustness interpretation of the STL specifications. Finally, the proposed termination conditions are based on the percentage of signals belonging to the same class associated to a node in the decision tree and the depth of the tree.

The decision tree is constructed recursively in three major steps:

- *Step 1:* check if the terminal conditions are satisfied on the current node.
- *Step 2:* if yes, create a leaf node, associate it to the label that gives the best classification and stop the current recursion iteration. If not, create a non-terminal node and associate it with the STL formula that best splits the data reaching that node.
- *Step 3:* partition the data that reach the current node according to whether they satisfy or violate its associate specification and build accordingly two sub-trees. Apply the same steps to each sub-tree.

Step 2 requires associating the new node to the STL formula that "best" splits the data. The STL formulas are chosen among all instantiations of PSTL specification templates defined by the user. The notion of best refers to the optimization of the impurity measure, also given by the user.

The remaining step consists in translating the binary decision tree to the overall STL formula. This is also done recursively on the structure of the tree. The leave nodes with positive (negative) labels are associated to the formula **true** (**false**). The STL formula associated to a tree with the STL formula φ in the root is the disjunction between two conjunctions: (i) the conjunction of φ and the STL formula associated to the left child sub-tree, and (ii) the conjunction between $\neg\varphi$ and the STL formula associated to the right child sub-tree. An example of the procedure is depicted in Fig. 9.

The decision tree generated by the above procedure can be in some cases very large with high classification accuracy on the training data. Such a tree may be over-fitting the training data and may not give explainable insights about the data. The dimensions of the constructed tree can be reduced using a pruning algorithm [18]. The main idea is to equip the tree with a cost that takes into account the percentage of mis-classified data and the width of the tree. A sequence of trees with decreasing size is constructed by varying the weights that are given to the mis-classification rate and the width of the tree in the cost. Finally, it is possible to choose among the obtained trees picking the one that has the minimum classification error on the validation set.

4.4.2. Online supervised learning

This section describes a procedure that extends the supervised mining approach based on decision trees [19] presented in Section 4.4.1 from offline to online learning [17]. In the online learning scenario, data is not fully available prior to training, but rather arrives continuously over time. The aim of the online mining procedure is an efficient on-the-fly generation and refinement of the STL specification according to the new observed data.

The overall structure of the mining procedure is similar to its offline counterpart. That is, the decision tree classifies the observed labeled signals. The candidate specifications belong to a set of primitive formulas and the combination thereof. Finally, an impurity measure is used to define a criterion for selecting the primitive formula that splits data well with respect to their labels.

The key difference of the online mining method is its mechanism to update the learned STL specification with the arrival of new signals. There is a trivial solution to the online mining problem that consists in applying the offline procedure at each new arrival of data. This solution is nevertheless highly inefficient and expensive.

The main difficulty for an efficient update of a decision tree happens when a new signal causes the change of the best primitive formula associated to a node, according to the used impurity measure. To make the online learning procedure efficient, decision tree updates of this kind are postponed to when the new STL formula is reasonably certain to hold in the future, a decision based on a probabilistic condition.

4.4.3. Offline unsupervised learning

This section presents a procedure for offline unsupervised mining of STL specifications using decision trees [16]. In contrast to the other approaches in this section, this procedure clusters signals according to their temporal logic properties and then describes and discriminates the groups by appropriate STL formulas. In this setting, the input data does not need to be labeled since the clustering problem is an unsupervised problem that require to work uniquely on raw signals. The output of the algorithm is a hierarchical tree where each node verifies if a temporal property of the data is satisfied or not and each leaf represents a cluster that is mapped to an STL formula.

The mining procedure is similar to its supervised offline counterpart. However, the interpretation of the decision tree is different. The decision tree provides a hierarchical representation of signal clusters. The initial node contains the entire set of signals. When moving down the tree hierarchy, each node presents a refined set of similar signals. The leaf nodes represent the actual signal clusters. Each node also has an associated STL specification that characterizes the set of signals represented by the node. This STL formula aims to maximize discrimination between the signals associated to that node, and all the other signals.

The discrimination is achieved using a *homogeneity* measure that quantifies the distance between two signals (e.g. Euclidean Distance or Dynamic Time Warping).

4.5. Mining STL specifications by enumeration

This section presents a procedure for learning STL specification classifiers by combining enumerative search of monotonic PSTL templates with robustness-based instantiation of PSTL parameters [54].

The procedure is summarized in Algorithm 1 and consists of two major interleaved steps:

- *Step 1:* systematic enumeration of PSTL templates ordered by their size (function *Enumerate* in line 4), and
- *Step 2:* logical classification by instantiation of each template to an STL formula by minimizing its mis-classification rate.

The above procedure stops as soon as an STL formula with low mis-classification rate is found. Due to the exploration of specification templates in the increasing order of their size, the resulting STL formula is guaranteed to be the smallest one that satisfies the criterion.

Logical classification for every PSTL specification in the enumeration is computationally expensive with the risk to repeat the parameter instantiation for many equivalent PSTL formulas. To detect and remove equivalent, and hence redundant PSTL formulas before their logical classification is performed, the notion of signature is used to approximately check PSTL specifications for equivalence. The signature of a PSTL formula corresponds to a matrix containing the robustness values with respect to different traces and parameter valuations (this is what function *ComputeSignature* computes in line 6). Finding the parameters of a PSTL formula that minimizes mis-classification rate is done using an adaptation of the binary search algorithm [6] that approximates the satisfaction boundary of the formula (function *SelectValuation* in line 16). An instance of the binary search algorithm is illustrated in Fig. 10.

The above procedure is adapted to learn the environment assumption in a CPS model [53]. Following the observation that if the input traces satisfy this assumption, then the corresponding output traces satisfy the output requirement, the aim is to learn an STL classifier that partitions the input signals into the ones that result in the satisfaction, and the others that result in the violation of the system requirements. The robustness-driven method to learn parameter values is replaced by a decision-tree based approach.

Algorithm 1: Enumerative mining approach presented in [54].

Input: Maximum length *max_length*, acceptable MCR δ, set of traces $\mathcal{S}$
Output: STL formula with low MCR (if it is found)

```
1  l ← 1
2  D ← ∅
3  while l ≤ max_length do
4      Φ ← Enumerate(l)
5      foreach φ ∈ Φ do
6          s_φ ← ComputeSignature(φ)
7      while Φ \ D ≠ ∅ do
8          pick φ ∈ Φ \ D
9          is_new ← true
10         foreach ψ ∈ D do
11             if s_φ = s_ψ then
12                 D ← D ∪ {φ}
13                 is_new ← false
14                 break
15         if is_new then
16             v ← SelectValuation(φ, S)
17             if MCR(φ_v) ≤ δ then
18                 return φ_v
19             else
20                 D ← D ∪ {φ}
21     l ← l + 1
22 return no STL formula found
```

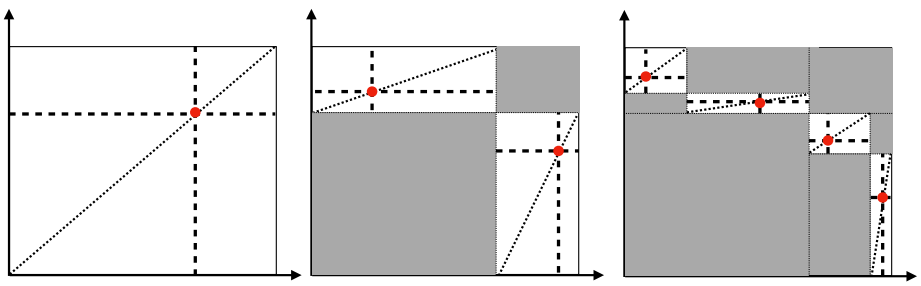

Fig. 10. By leveraging the monotonicity of the PSTL formula, the adaptation of the binary search algorithm consists in recursively dividing the parameter space into sub-regions containing the formula's satisfaction boundary. Each red point is the valuation on the diagonal of the respective hyper-box that instantiates an STL formula with robustness value equal to zero. Depending on the monotonicity direction, two of the generated sub-regions are excluded from further search (gray boxes) because they only contain either all valid or invalid valuations, hence no points on the satisfaction boundary. The procedure is then iteratively repeated on the remaining sub-regions.

4.6. Mining STL specifications from positive examples guided by robustness metrics

This section presents an incremental procedure for passively mining STL specifications from positive examples. This procedure extends the approach that infers parameters in STL specification templates guided by tightness metrics [35] presented in Section 3.3 with the ability to also learn the specification structure from data [36]. The main objective is to learn the tightest STL formula that is satisfied by all positive examples.

The core of the method is to build the formula structure incrementally, constructing in each iteration a more complex STL formula. The overall approach is summarized in Algorithm 2. The algorithm takes as input a set of traces and a set of candidate PSTL predicate templates (i.e. parametric inequalities over the signals such as $f(x) \geq k$, where x is a signal and k a parameter). The procedure is iterative, and in each iteration it updates a set Ψ of candidate specifications. This set is initiated with simple formulas of the form $\mathbf{F}_I p$, $\mathbf{G}_I p$ and $p\mathbf{U}_I q$, where p and q are PSTL predicate templates and $I = [a, b]$ is a parameterized time interval (function *InitializeSet* in line 1). An iteration of the procedure consists of the following steps:

- *Step 1:* apply the parameter inference procedure guided by the tightness metric to all the PSTL specifications in the set Ψ (function *ParameterInference* in line 5) and create the corresponding set of STL formulas Φ.
- *Step 2:* filter the resulting STL formulas according to their robustness w.r.t. the positive examples (function *Filter* in line 7). The filtering uses a selection heuristic, which initially keeps the set of formulas diverse (keeping even some formulas with negative robustness), then it gradually favors only formulas which are tightly satisfied by the set of traces.

- *Step 3:* if no STL formula remains in Φ after filtering, the procedure is not able to find an STL specification that is consistent with the positive examples and the procedure stops.
- *Step 4:* if there is only one remaining STL formula in Φ after filtering, the procedure returns this as output specification and stops.
- *Step 5:* augment the set Ψ by nesting existing sub-formulas with another temporal ($\mathbf{F}_I$, $\mathbf{G}_I$ or $\mathbf{U}_I$) operator, and combining the existing sub-formulas with disjunction (function *Augment* in line 13).

Algorithm 2: Mining procedure in [36].

```
   Input: Set of traces S, set of PSTL predicate templates P
   Output: STL formula (if it is found)
 1 Ψ ←InitializeSet(P)
 2 while true do
 3     Φ ← ∅
 4     foreach φ ∈ Ψ do
 5         φ_v ← ParameterInference(φ)
 6         Φ ← Φ ∪ {φ_v}
 7     Φ ←Filter(Φ)
 8     if |Φ| = 0 then
 9         return no STL formula found
10     else if |Φ| = 1 then
11         return φ_v ∈ Φ
12     else
13         Ψ ←Augment(Ψ)
```

4.7. Mining STL specifications with genetic algorithms

Genetic algorithms can be used to tackle the problem of automatically inferring an STL formula that acts as a two-class classifier among two sets of signals, giving also insights on what distinguishes the two sets. The traces in the training data set needs to have been previously labeled as either desired or anomalous trajectories of the system under study.

Genetic algorithms take inspiration from the process of natural selection among species. In the specification mining applications, the term *individual* refers to a candidate formula. The definition of three aspects characterizes a particular genetic algorithm: 1) fitness function (indicating how an individual is suitable to survive, i.e., how good a formula discriminates between the sets of positive and negative examples), 2) crossover operator (how a new individual is generated starting from two others, i.e., how formulas are combined to have new ones), and 3) mutation operator (what random changes may happen to an individual formula).

There are two similar approaches that use genetic algorithms for mining STL formulas. The first approach is a supervised learning algorithm for mining an STL specification classifier by combining an Evolutionary Algorithm (EA) to learn the structure of the formula with a Gaussian Process Upper Confidence Bound Algorithm (GP-UCB) to infer its optimal parameters [59]. The approach in [59] considers the full STL specification language and not just specific fragments. Another genetic algorithm to mine specifications is presented in [68,7]. The second approach is a pure genetic algorithm approach and restricts its attention to the past STL (ptSTL) fragment, which forbids the use of future temporal operators. Since these two approaches are similar, we sketch in more detail the former approach only.

The procedure takes as input two labeled datasets: the set of positive and the set of negative examples. The procedure is described in Algorithm 3 and starts with the generation of a set of templates via conjunction of predicates or application of temporal operators to predicates (function *InitializeTemplateSet* in line 1). The rest of the algorithm consists of the structure mining and the parameter inference steps, which are combined in a compositional manner. In every iteration of the procedure:

- *Step 1:* EA updates the set of candidate specification templates,
- *Step 2:* for each specification template, GP-UCB is used to find the parameters that optimize its discrimination between the sets of positive and negative examples.

These two steps require four main ingredients: (i) the fitness function, (ii) the discrimination function, (iii) the cross-over operation, and (iv) the mutation operator.

The fitness function is used to sample the formulas from the set of candidate templates that will generate new formulas or will be passed to the following iteration (function *Sample* in lines 4 and 7, respectively). This function is defined as the sum of two terms: one that penalizes long-size formulas and the other that favors formulas with higher discrimination power. The latter is defined as a *discrimination* function that captures how well an STL formula discriminates between two sets of examples. It assigns to the formula the difference between the average robustness of positive and the average

Algorithm 3: Genetic algorithm to mine specifications [59].

Input: Set of positive examples $\mathcal{S}_p$, set of negative examples $\mathcal{S}_n$, fitness function F, discrimination function D, maximum number of iterations N
Output: STL formula discriminating $\mathcal{S}_p$ from $\mathcal{S}_n$

1 $\Psi \leftarrow$InitializeTemplateSet
2 $\Phi \leftarrow$ ParameterInference($\Psi, D, \mathcal{S}_p, \mathcal{S}_n$)
3 **for** $i = 1, \ldots, N$ **do**
4 $\quad \Phi_s \leftarrow$Sample($\Phi, F, \mathcal{S}_p, \mathcal{S}_n$)
5 $\quad \Psi \leftarrow$ Crossover(Φ_s) *or* Mutation(Φ_s)
6 $\quad \Phi_s \leftarrow$ ParameterInference($\Psi, D, \mathcal{S}_p, \mathcal{S}_n$)
7 $\quad \Phi \leftarrow$Sample($\Phi \cup \Phi_s, F, \mathcal{S}_p, \mathcal{S}_n$)
8 $\varphi \leftarrow$ SelectFormula(Φ)
9 **return** φ

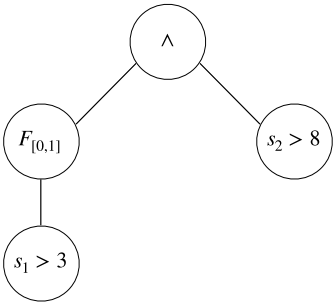

Fig. 11. Syntax tree of the formula $\varphi = \mathbf{F}_{[0,1]}(s_1 > 3) \wedge (s_2 > 8)$.

robustness of negative examples, divided by the sum of the respective standard deviations. This is the function that GB-UCP aims to maximize in order to find the optimal parameters for each formula template (function *ParameterInference* in lines 2 and 6).

Considering the representation of a formula as a syntax tree (see an example in Fig. 11), the cross-over operator works as follows: given two formulas (the parents), it randomly selects one subtree from each formula and switches them, generating two new formulas (the children). Finally, the mutation operator takes as input a formula and it changes one random node with one random formula (either predicates or temporal operators **F**, **G** and **U** applied to predicates). At each iteration, the cross-over operator and the mutation operator are mutually exclusive and randomly selected according to their predefined probabilities (line 5).

After a maximum number of iterations, the algorithm returns the formula in the set of candidate specifications that have the maximum discrimination power according to the discrimination function (function *SelectFormula* in line 8).

5. Related work

This tutorial focuses on revisiting methods for learning formal specifications in Signal Temporal Logic [51] (STL). However, there are other approaches to learn different formal specification languages and models. In the following, we briefly report some of them. For a more general overview we refer the reader to a recent interesting survey [77].

Multi-valued specification languages STL with quantitative semantics is a multi-valued logic assuming values in $\mathbb{R} \cup \{-\infty, \infty\}$ and allowing to measure how far is a behavior from satisfying or violating a specification. There are other branches of multi-valued logic such as fuzzy logic [80] and probabilistic logic [64]. In contrast to STL, fuzzy logic is normalized to the interval $[0, 1]$ of truth values and aims to represent vagueness and imprecise information. Probabilistic logic allows to reason with uncertain knowledge by having the truth values of sentences be probabilities between 0 and 1.

Learning invariants Temporal invariants are properties that remain unchanged during the system's executions. In STL such property would correspond to learn the template $\mathbf{G}\varphi$, where φ is a constraint expressed over the system's variables (for example $x \leq y + 1$) that the system will always satisfy. A popular tool to learn such invariants is Daikon [25] and it is employed in CPSDebug [15] to support failure explanation in CPS models.

Learning LTL LTL describes temporal properties of reactive systems. Texada [48] is a popular approach for learning LTL specifications from temporal templates by instantiating the template with the propositions that are consistent with the observed data. Huang and Cleaveland [34] propose an automata-based approach for inferring LTL formula from a specification template and a set of data streams. Neider and Gavran [57] introduce two passive approaches for learning LTL - the first one consists of a sequence of satisfiability problems in propositional Boolean logic that generate the smallest LTL specification that is consistent with the observations, while the second one combines the SAT-driven learning with learning decision

trees. The second approach allows to scale to large-size scenarios while dropping the minimality requirement. Riener [67] improves on the SAT-based learning method by using topology structures to partition the search space into small sub-problems. Gaglione et al. [30] extend the LTL mining method from Neider and Gavran by enabling inference of concise LTL specifications in the presence of noisy data by using MaxSMT.

Learning state-machines State-machine models such as finite state automata provide an alternative approach to express the system's behavior. They consist of a finite set of states and a transition function determining how to switch from a state to another given a certain input symbol. Dana Angluin proposed in [3] the first learning method for deterministic finite automata called L^* algorithm. This algorithm can be formulated as a cooperative game between two players: a learner and a teacher. The aim of the learner is to infer, by formulating some queries and reasoning about the obtained answers, a deterministic finite automaton modeling an unknown language L. The teacher acts instead as an oracle that can answer queries formulated by the learner. There are generally two types of queries: *membership* and *equivalence* queries. The first type of queries consists in asking whether a certain word belongs to the unknown language, while the second type of queries asks whether a candidate language is equivalent to the unknown language. The family of approaches based on this interaction between a learner and a teacher are also called *active learning methods* [71,32,77]. The main prerequisite of these methods is the existence of a teacher, but this is not always feasible. An alternative approach that does not require a teacher is given in ALERGIA algorithm [21] where a prefix tree acceptor is built starting from a set of observed execution traces. Approaches that do not require the interaction with a teacher are called *passive learning methods* [77]. More recently, many of these concepts have been further extended to learn more expressive and complex state machine models such as timed automata [65], linear hybrid automata [70,52], and hybrid automata with inputs and outputs [52].

Learning regular expressions *Regular expressions* describe the regular languages accepted by deterministic finite automata and can be used to specify in a concise way patterns of symbols. A regular expression can be automatically generated from a deterministic finite automata using Kleene's algorithm [31,44]. A method for mining timed regular expressions [5], from traces was developed in [55,56]. *Shape expressions* [62] are regular expressions where the symbols are parametrized functions (linear, exponential, sinosuoidal, etc) describing shapes that characterize the features of a signal. Recent works [10,11] provide an automated approach to learn linear shape expressions from a set of positive examples and it is supported by ShapeIt [11].

Learning model checking Learning model checking is a new interesting framework introduced in [20] that consists in predicting the probability of satisfaction (or the robustness value) of an STL specification with respect to a model M, given the satisfaction values of a set of independently selected STL properties. The key insight is to learn a linear model that predicts the satisfaction values of STL formulas in a feature space, namely a higher dimensional space than the original space of data and where it is easier to work on. As commonly used in machine learning, the kernel trick is exploited to perform an efficient learning that does not require the explicit computation of the feature space: it is indeed sufficient to use the similarity between pairs of STL formulas. The similarity between two formulas is defined as the scalar product of the functions that compute the robustness values of the formulas by varying the trajectories. The authors mention the possibility of using their new approach for specification mining purposes: the search would then be shifted from the discrete space of formula templates to a continuous space where distances depend on semantic similarities.

Data mining in temporal series Mining patterns in temporal series has been the subject of intense and prolific research activities for at least three decades. As discussed in [41] the main tasks in temporal series data mining are *indexing*, *clustering*, *classification* and *segmentation*. *Indexing* [40] consists in matching times series in a database that are the closest according to a similarity measure to a given query time series. *Clustering* [40,43] is a task that groups times series according to a similarity measure, while *classification* [40,43] is a supervised machine learning task that takes in input an unlabeled time series and determines one or more classes to which the time series belongs. Finally the goal of *segmentation* [42,69] is to compute from an initial time series, another time series with less data points approximating the original one. This last task is general functional to efficiently perform the other three. Mining STL specifications can be considered as *clustering* and *classification* tasks, while monitoring an STL formula over time series is similar to the *indexing* task. However, one main difference is that we can leverage a data-efficient, concise and human-understandable symbolic language to represent the sets of time series we are interested in. This provides a suitable tool not only to classify/cluster time series, but also to explain the temporal relations characterizing the essence of the desired time series pattern. Furthermore, the use of STL opens to the possibility of using all the machinery available for monitoring, testing, and control synthesis.

6. Conclusion and future directions

This paper evidences the active and vivid research in the domain of mining STL specifications. This field connects the worlds of machine learning and formal methods, exploiting many potential synergies between the two. We observe that both template-based and template-free mining approaches have relevant practical applications and receive considerable attention from the research community. Despite many different angles used to tackle this problem, we see several unexplored opportunities for research, that we briefly discuss in this section.

Model-based active mining We notice that only one reviewed approach uses the system model in the specification mining process. We think that this is an interesting and not sufficiently explored direction to pursue, especially when using specification mining to support the CPS design. More specifically, the model can be used to generate new behaviors and thus actively refine the mined specification, providing in every iteration a more precise insight in the model properties.

Mining input/output relations All the approaches considered in this paper assume that the outcome is a flat specification that characterizes a language. In CPS, the systems are dominantly *open* and have a clear input and output signature – the system continuously receives inputs and reacts to them, generating outputs. For many applications, mining an input/output relation would be a very valuable outcome.

Mining local patterns The methods reviewed in this paper focus on learning specifications from the entire signals. A different view on the mining problem consists in finding repeating anomalous patterns within a single behavior. This more local mining problem has been extensively studied in the time series community with the ideas of shapelets [72] and motifs [79] but has received no attention so far in the context of STL.

Explainable AI Specification mining is a form of explainable artificial intelligence, since a formal specification provides a-priori an interpretable artefact. However, this corner on learning specifications has not been sufficiently studied, and there is little work that investigates, for instance, whether STL specifications can provide meaningful additional insight into opaque black-box components, such as deep neural networks.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Dejan Nickovic, Cristinel Mateis, Eleonora Nesterini reports financial support was provided by European Commission.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123 and it is partially funded by the TU Wien-funded Doctoral College SecInt: Secure and Intelligent Human-Centric Digital Technologies. The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

References

[1] C. Ackermann, R. Cleaveland, S. Huang, A. Ray, C.P. Shelton, E. Latronico, Automatic requirement extraction from test cases, in: Proc. of RV 2010, in: LNCS, vol. 6418, Springer, 2010, pp. 1–15.
[2] Rajeev Alur, Tomás Feder, Thomas A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.
[3] Dana Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (2) (1987) 87–106.
[4] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, Sriram Sankaranarayanan, S-TaLiRo: a tool for temporal logic falsification for hybrid systems, in: Proc. of TACAS 2011, in: LNCS, vol. 6605, Springer, 2011, pp. 254–257.
[5] Eugene Asarin, Paul Caspi, Oded Maler, Timed regular expressions, J. ACM 49 (2) (2002) 172–206.
[6] Eugene Asarin, Alexandre Donzé, Oded Maler, Dejan Nickovic, Parametric identification of temporal properties, in: Proc. of RV 2011, in: LNCS, vol. 7186, Springer, 2011, pp. 147–160.
[7] Sertaç Kagan Aydin, Ebru Aydin Gol, Synthesis of monitoring rules with STL, J. Circuits Syst. Comput. 29 (11) (2020) 2050177.
[8] Ezio Bartocci, Luca Bortolussi, Michele Loreti, Laura Nenzi, Simone Silvetti MoonLight, A lightweight tool for monitoring spatio-temporal properties, in: Proc. of RV 2020, in: LNCS, vol. 12399, Springer, 2020, pp. 417–428.
[9] Ezio Bartocci, Luca Bortolussi, Guido Sanguinetti, Data-driven statistical learning of temporal logic properties, in: Proc. of FORMATS 2014, in: LNCS, vol. 8711, Springer, 2014, pp. 23–37.
[10] Ezio Bartocci, Jyotirmoy Deshmukh, Felix Gigler, Cristinel Mateis, Dejan Nickovic, Xin Qin, Mining shape expressions from positive examples, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3809–3820.
[11] Ezio Bartocci, Jyotirmoy Deshmukh, Cristinel Mateis, Eleonora Nesterini, Dejan Nickovic, Xin Qin, Mining shape expressions with ShapeIt, in: Proc. of SEFM 2021, in: LNCS, vol. 13085, Springer, 2021, pp. 110–117.
[12] Ezio Bartocci, Jyotirmoy V. Deshmukh, Alexandre Donzé, Georgios E. Fainekos, Oded Maler, Dejan Nickovic, Sriram Sankaranarayanan, Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications, in: Lectures on Runtime Verification - Introductory and Advanced Topics, in: LNCS, vol. 10457, Springer, 2018, pp. 135–175.
[13] Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, Dejan Nickovic, Localizing faults in Simulink/Stateflow models with STL, in: Proc. of HSCC 2018, ACM, 2018, pp. 197–206.
[14] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Nickovic, Automatic failure explanation in CPS models, in: Proc. of SEFM 2019, in: LNCS, vol. 11724, Springer, 2019, pp. 69–86.
[15] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Nickovic, CPSDebug: Automatic failure explanation in CPS models, Int. J. Softw. Tools Technol. Transf. 23 (5) (2021) 783–796.
[16] Giuseppe Bombara, Calin Belta, Signal clustering using temporal logics, in: Proc. of RV 2017, in: LNCS, vol. 10548, Springer, 2017, pp. 121–137.
[17] Giuseppe Bombara, Calin Belta, Online learning of temporal logic formulae for signal classification, in: Proc. of ECC 2018, IEEE, 2018, pp. 2057–2062.
[18] Giuseppe Bombara, Calin Belta, Offline and online learning of signal temporal logic formulae using decision trees, ACM Trans. Cybern. Phys. Syst. 5 (3) (2021) 22.

[19] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, Calin Belta, A decision tree approach to data classification using signal temporal logic, in: Proc. of HSCC 2016, ACM, 2016, pp. 1–10.
[20] Luca Bortolussi, Giuseppe Maria Gallo, Jan Křetínský, Laura Nenzi, Learning model checking and the kernel trick for signal temporal logic on stochastic processes, in: Proc. of TACAS 2022, in: LNCS, vol. 13243, Springer, 2022, pp. 281–300.
[21] Rafael C. Carrasco, José Oncina, Learning stochastic regular grammars by means of a state merging method, in: Proc. of ICGI 94, in: LNCS, vol. 862, Springer, 1994, pp. 139–152.
[22] Yushan Chen, Jana Tumova, Alphan Ulusoy, Calin Belta, Temporal logic robot control based on automata learning of environmental dynamics, Int. J. Robot. Res. 32 (5) (2013) 547–565.
[23] Alexandre Donzé Breach, A toolbox for verification and parameter synthesis of hybrid systems, in: Tayssir Touili, Byron Cook, Paul Jackson (Eds.), Proc. of CAV 2010, in: LNCS, vol. 6174, Springer, 2010, pp. 167–170.
[24] Alexandre Donzé, Oded Maler, Robust satisfaction of temporal logic over real-valued signals, in: Proc. of FORMATS 2010, in: LNCS, vol. 6246, Springer, 2010, pp. 92–106.
[25] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, Chen Xiao, The Daikon system for dynamic detection of likely invariants, Sci. Comput. Program. 69 (1–3) (2007) 35–45.
[26] François Fages, Aurélien Rizk, From model-checking to temporal logic constraint solving, in: Proc. of CP 2009, in: LNCS, vol. 5732, Springer, 2009, pp. 319–334.
[27] Georgios E. Fainekos, George J. Pappas, Robustness of temporal logic specifications, in: Proc. of FATES 2006 and RV 2006, in: LNCS, vol. 4262, Springer, 2006, pp. 178–192.
[28] Georgios E. Fainekos, George J. Pappas, Robustness of temporal logic specifications for continuous-time signals, Theor. Comput. Sci. 410 (42) (2009) 4262–4291.
[29] Jeanne Ferrante, Charles Rackoff, A decision procedure for the first order theory of real addition with order, SIAM J. Comput. 4 (1) (1975) 69–76.
[30] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, Zhe Xu, Learning linear temporal properties from noisy data: a MaxSAT-based approach, in: Proc. of ATVA 2021, in: LNCS, vol. 12971, Springer, 2021, pp. 74–90.
[31] John E. Hopcroft, Jeff D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, 1979.
[32] Falk Howar, Bernhard Steffen, Active automata learning in practice, in: Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, in: LNCS, vol. 11026, Springer, 2018, pp. 123–148.
[33] Bardh Hoxha, Adel Dokhanchi, Georgios E. Fainekos, Mining parametric temporal logic properties in model-based design for cyber-physical systems, Int. J. Softw. Tools Technol. Transf. 20 (1) (2018) 79–93.
[34] Samuel Huang, Rance Cleaveland, Temporal-logic query checking over finite data streams, in: Proc. of FMICS 2020, in: LNCS, vol. 12327, Springer, 2020, pp. 252–271.
[35] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, Natarajan Shankar TeLEx, Passive STL learning using only positive examples, in: Proc. of RV 2017, in: LNCS, vol. 10548, Springer, 2017, pp. 208–224.
[36] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, Natarajan Shankar, TeLEx: learning signal temporal logic from positive examples using tightness metric, Form. Methods Syst. Des. 54 (3) (2019) 364–387.
[37] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, Sanjit A. Seshia, Mining requirements from closed-loop control models, in: Proc. of HSCC 2013, ACM, 2013, pp. 43–52.
[38] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, Sanjit A. Seshia, Mining requirements from closed-loop control models, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (11) (2015) 1704–1717.
[39] Austin Jones, Zhaodan Kong, Calin Belta, Anomaly detection in cyber-physical systems: a formal methods approach, in: Proc. of CDC 2014, IEEE, 2014, pp. 848–853.
[40] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, Michael J. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, in: Proc. of ACM SIGMOD 2001, ACM, 2001, pp. 151–162.
[41] Eamonn J. Keogh, Shruti Kasetty, On the need for time series data mining benchmarks: a survey and empirical demonstration, Data Min. Knowl. Discov. 7 (4) (2003) 349–371.
[42] Eamonn J. Keogh, Michael J. Pazzani, An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback, in: Proc. of KDD 98, AAAI Press, 1998, pp. 239–243.
[43] Dennis F. Kibler, Pat Langley, Machine learning as an experimental science, in: Proc. of EWSL 1988, Pitman Publishing, 1988, pp. 81–92.
[44] S.C. Kleene, Representation of Events in Nerve Nets and Finite Automata, RAND Corporation, Santa Monica, CA, 1951.
[45] Zhaodan Kong, Austin Jones, Calin Belta, Temporal logics for learning and detection of anomalous behavior, IEEE Trans. Autom. Control 62 (3) (2017) 1210–1222.
[46] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, Calin Belta, Temporal logic inference for classification and prediction from data, in: Proc. of HSCC 2014, ACM, 2014, pp. 273–282.
[47] Ron Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4) (1990) 255–299.
[48] Caroline Lemieux, Dennis Park, Ivan Beschastnikh, General LTL specification mining (T), in: Proc. of ASE 2015, IEEE, 2015, pp. 81–92.
[49] Karen Leung, Nikos Arechiga, Marco Pavone, Back-propagation through signal temporal logic specifications: infusing logical structure into gradient-based methods, in: Proc. of WAFR 2021, in: Springer Proceedings in Advanced Robotics, vol. 17, Springer, 2020, pp. 432–449.
[50] Rüdiger Loos, Volker Weispfenning, Applying linear quantifier elimination, Comput. J. 36 (5) (1993) 450–462.
[51] Oded Maler, Dejan Nickovic, Monitoring temporal properties of continuous signals, in: Proc. of FORMATS 2004 and FTRTFT 2004, in: LNCS, vol. 3253, Springer, 2004, pp. 152–166.
[52] Ramy Medhat, S. Ramesh, Borzoo Bonakdarpour, Sebastian Fischmeister, A framework for mining hybrid automata from input/output traces, in: Proc. of EMSOFT 2015, IEEE, 2015, pp. 177–186.
[53] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Mining environment assumptions for cyber-physical system models, in: Proc. of ICCPS 2020, IEEE, 2020, pp. 87–97.
[54] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Marcell Vazquez-Chanlatte, Alexandre Donzé, Interpretable classification of time-series data using efficient enumerative techniques, in: Proc. of HSCC 2020, ACM, 2020, 9.
[55] Apurva Narayan, Greta Cutulenco, Yogi Joshi, Sebastian Fischmeister, Mining timed regular specifications from system traces, ACM Trans. Embed. Comput. Syst. 17 (2) (2018) 46.
[56] Apurva Narayan, Sebastian Fischmeister, Mining time for timed regular specifications, in: Proc. of SMC 2019, IEEE, 2019, pp. 63–69.
[57] Daniel Neider, Ivan Gavran, Learning linear temporal properties, in: Proc. of FMCAD 2018, IEEE, 2018, pp. 1–10.
[58] Laura Nenzi, Ezio Bartocci, Luca Bortolussi, Simone Silvetti, Michele Loreti MoonLight, A lightweight tool for monitoring spatio-temporal properties, in: Proc. of RV 2020, in: LNCS, vol. 12399, Springer, 2020, pp. 417–428.
[59] Laura Nenzi, Simone Silvetti, Ezio Bartocci, Luca Bortolussi, A robust genetic algorithm for learning temporal specifications from data, in: Proc. of QEST 2018, in: LNCS, vol. 11024, Springer, 2018, pp. 323–338.
[60] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancic, Aarti Gupta, George J. Pappas, Monte-Carlo techniques for falsification of temporal properties of non-linear hybrid systems, in: Proc. of HSCC 2010, ACM, 2010, pp. 211–220.

[61] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, Ken Butts, Taylor T. Johnson, Abnormal data classification using time-frequency temporal logic, in: Proc. of HSCC 2017, ACM, 2017, pp. 237–242.
[62] Dejan Nickovic, Xin Qin, Thomas Ferrère, Cristinel Mateis, Jyotirmoy Deshmukh, Specifying and detecting temporal patterns with shape expressions, Int. J. Softw. Tools Technol. Transf. 23 (4) (2021) 565–577.
[63] Nickovic Dejan, Tomoya Yamaguchi, RTAMT: online robustness monitors from STL, in: Proc. of ATVA 2020, in: LNCS, vol. 12302, Springer, 2020, pp. 564–571.
[64] Nils J. Nilsson, Probabilistic logic, Artif. Intell. 28 (1) (1986) 71–87.
[65] Fabrizio Pastore, Daniela Micucci, Leonardo Mariani, Timed k-tail: automatic inference of timed automata, in: Proc. of ICST 2017, IEEE, 2017, pp. 401–411.
[66] Amir Pnueli, The temporal logic of programs, in: Proc. of SFCS 1977, IEEE, 1977, pp. 46–57.
[67] Riener Heinz, Exact synthesis of LTL properties from traces, in: Proc. of FDL 2019, IEEE, 2019, pp. 1–6.
[68] Irmak Saglam, Ebru Aydin Gol, Cause mining and controller synthesis with STL, in: Proc. of CDC 2019, IEEE, 2019, pp. 4589–4594.
[69] Hagit Shatkay, Stanley B. Zdonik, Approximate queries and representations for large data sequences, in: Proc. of the ICDE 1996, IEEE, 1996, pp. 536–545.
[70] Miriam García Soto, Thomas A. Henzinger, Christian Schilling, Luka Zeleznik, Membership-based synthesis of linear hybrid automata, in: Proc. of CAV 2019, in: LNCS, vol. 11561, Springer, 2019, pp. 297–314.
[71] Bernhard Steffen, Falk Howar, Maik Merten, Introduction to active automata learning from a practical perspective, in: Proc. of SFM 2011, in: LNCS, vol. 6659, Springer, 2011, pp. 256–296.
[72] Liudmila Ulanova, Nurjahan Begum, Eamonn J. Keogh, Scalable clustering of time series with U-Shapelets, in: Proc. of SDM 2015, SIAM, 2015, pp. 900–908.
[73] Ulus Dogan, Online monitoring of metric temporal logic using sequential networks, CoRR, arXiv:1901.00175 [abs], 2019.
[74] Prashant Vaidyanathan, Rachael Ivison, Giuseppe Bombara, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, Calin Belta, Grid-based temporal logic inference, in: Proc. of CDC 2017, IEEE, 2017, pp. 5354–5359.
[75] Marcell Vazquez-Chanlatte, Jyotirmoy V. Deshmukh, Xiaoqing Jin, Sanjit A. Seshia, Logical clustering and learning for time-series data, in: Proc. of CAV 2017, in: LNCS, vol. 10426, Springer, 2017, pp. 305–325.
[76] Marcell Vazquez-Chanlatte, Shromona Ghosh, Jyotirmoy V. Deshmukh, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, Time-series learning using monotonic logical properties, in: Proc. of RV 2018, in: LNCS, vol. 11237, Springer, 2018, pp. 389–405.
[77] Fujun Wang, Zining Cao, Lixing Tan, Hui Zong, Survey on learning-based formal methods: taxonomy, applications and possible future directions, IEEE Access 8 (2020) 108561–108578.
[78] Hengyi Yang, Bardh Hoxha, Georgios E. Fainekos, Querying parametric temporal logic properties on embedded systems, in: Proc. of ICTSS 2012, in: LNCS, vol. 7641, Springer, 2012, pp. 136–151.
[79] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, Eamonn J. Keogh, Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets, in: Proc. of ICDM 2016, IEEE, 2016, pp. 1317–1322.
[80] Lotfi A. Zadeh, Fuzzy logic, Computer 21 (4) (1988) 83–93.