Active Learning of Signal Temporal Logic Specifications

Alexis Linard and Jana Tumova

Abstract— **In this paper, we propose a method to infer temporal logic behaviour models of an *a priori* unknown system. We use the formalism of Signal Temporal Logic (STL), which can express various robot motion planning and control specifications, including spatial preferences. In our setting, data is collected through a series of queries the learning algorithm poses to the system under test. This active learning approach incrementally builds a hypothesis solution which, over time, converges to the actual behaviour of the system. Active learning presents several benefits compared to supervised learning: in the case of costly prior labelling of data, and if the system to test is accessible, the learning algorithm can interact with the system to refine its guess of the specification of the system. Inspired by mobile robot navigation tasks, we present experimental case studies to ensure the relevance of our method.**

I. Introduction

Robot motion planning and control via formal methods have recently received increasing interest. Among others, temporal logics are widely used [1]. Such logics can express robotic tasks and can allow the specification of complex missions, including combinations of safety requirements, surveillance, sequencing and implications tasks. For instance, Linear Temporal Logic (LTL) allows rich expressiveness and interpretability in natural language, and has been identified as a suitable specification language for control and safety objectives [2]. Signal Temporal Logic (STL) is, for its part, defined on continuous signals and can be used for specifications on systems [3]. It can express system properties that include time bounds and bounds on physical system parameters. One of its advantages, compared to LTL, is its richness to specify continuous-time systems. It is, in the case of robotics, an attractive formalism to model, for instance, classes of desired trajectories.

Typically, system properties expressed through LTL or STL are designed based on domain or task knowledge, for tasks spanning control synthesis or even model checking. However, whenever there does not already exist a formal representation of the behaviour of the systems, the completion of motion planning tasks or the calculation of safety guarantees becomes intractable. Temporal Logic Inference methods aim at synthesizing behaviour descriptions from system data. They involve supervised machine learning problems, where the task is to classify observed behaviour as either satisfying or violating a temporal logic specification. In the case of data in the form of finite time series of signals, the task becomes to infer an STL formula optimally separating a set of signals, labelled as either satisfying (positive signals) or violating (negative signals). However, Temporal Logic Inference methods have typically considered a specific setting so far: data is collected and labelled before the inference task begins. That is, they do not deal with the lack of data before the learning task.

Active learning is a particular case in machine learning since learning algorithms have access to an information source able to guide the guess of a solution. The learning algorithm can, therefore, interactively query this information source in an iterative process. In this paper, we propose an active learning iterative algorithm that queries a System Under Test (SUT) from which we want to abstract an STL specification of its behaviour. The type of *system* from which we want to model the behaviour and that we consider in this paper has several properties. It has to be capable to answer a variety of questions concerning accepted behaviours and even concerning the semantics of its formal specification. If not directly, an *oracle* having such a piece of knowledge about the SUT should be able to answer these questions.

Our learning algorithm is in possession of a hypothesis specification of the behaviour of the SUT at any iteration. We define two ways the learning algorithm can dialogue with the SUT: the first consists of *membership queries*, where a generated signal (according to the hypothesis specification at the moment) is tested against the SUT, which will label the signal accordingly. The second consists of *equivalence queries*, where the learning algorithm asks whether its actual guessed specification so far complies with the SUT's implemented behaviour. If so, the algorithm returns its hypothesis, equivalent to the desired specification to learn; otherwise, the SUT provides the algorithm with a labelled counterexample. In our case, we perform testing of equivalence by retrieving the distance [4] between the hypothesis specification and the target specification. Our learning algorithm keeps updating its hypothesis using an online decision tree learning algorithm [5], where the decision tree tries to best separate the labelled signals observed so far. The decision tree consists of nodes containing a simple STL formula locally separating data, where the formula is chosen among a set of simple STL templates called *primitives*, and where an optimization process finds the best temporal and spatial parameters of the simple formula to maximize accuracy w.r.t. the data retrieved so far. The hypothesis specification is then obtained by recursively parsing the tree's nodes. During the process, we experimentally observed that the hypothesis specification converges towards the behaviour of the SUT.

Note that for many classes of systems, e.g. discrete-

The authors are with the KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden, with the division of Robotics, Perception and Learning, Faculty of Electrical Engineering and Computer Science. This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and the Swedish Research Council (VR). {linard,tumova}@kth.se

time systems, the answer to the *equivalence queries* can be retrieved through model checking [6]. Therefore, the instantiation of the *oracle* as an interface between the learning algorithm and the SUT is, in real-world applications, a realistic scenario. In the rest of the paper, we consider the hypothetical case where the system can directly answer the queries.

This paper is organized as follows: in Section II, we present the related work on learning temporal logic specifications from data. In Section III, we present definitions on STL as well as related concepts. In Section IV and V, we state our problem and detail our approach. Then, we present in Section VI our experimental case studies ensuring the relevance of our methods, and finally conclude and discuss future work in Section VII.

II. Related Work

Learning of specifications for systems is not a new task: temporal logic inference of LTL properties has been explored in [7], where two methods are proposed, one based on SAT solving, and the other on decision trees. More specifically, learning of STL properties has also been investigated recently in [8]. Literature can be divided into two parts.

The first part focuses on parameter synthesis given a PSTL formula, i.e. a parametric version of STL [9], [10]. The task is to learn the parameters of a PSTL formula such that the parameterized PSTL formula will correctly classify signals. This is typically done from a fully labelled dataset (i.e. having signals which are known to be satisfying the specification, and others are known to violate it), or from positive signals only [11]. For instance, in [12], the search of parameters is counterexample based, using a falsification algorithm given a Simulink model of the system. In [13], the authors focus on rPSTL (Reactive Parametric Signal Temporal Logic), a fragment of STL capturing causal relationships between cause-effect formulae. The authors of [14] focus on another fragment of STL, that is, iPSTL (inference Parametric Signal Temporal Logic, a fragment of PSTL and generalization of rPSTL). Finally, in [11], the authors come up with an algorithm for learning the parameters of a PSTL formula from positive signals only, i.e. where signals which are known to satisfy the formula are available, but signals violating the formula are lacking.

The second part surveys both building the STL formula itself, as well as mining its parameters. Concerning supervised learning, we can cite, for instance, [15], which employs an evolutionary algorithm to learn an STL formula from positive and negative signals. In a different fashion, the authors of [16] proposed a decision tree-based method to learn an STL formula. The nodes of the decision tree contain a test with the satisfaction of a simple STL formula called *primitive*, in order to separate negative and positive signals in a dataset. The decision tree represents the learnt STL formula. The same authors also proposed in [5] an online version of the decision tree method, where the decision tree is updated on-the-fly whenever new instances of labelled signals are available. Unsupervised learning approaches have also been developed in [17]. The authors there consider the inference of STL formulae from data, based on an adaptation of [13] to a one-class SVM based optimization function, in the context of the prediction of anomalies of Cyber-Physical Systems. In [18], the authors perform unsupervised learning of the whole class of STL formulae from data. The algorithm proposed is grid-based (projection of signals to a grid), and the finer the grid is, the finer would be the clustering. Then, for each cluster, a formula is inferred. Finally, in [19], the method proposed first performs the discovery of PSTL templates to group signals with similar logical properties together, using multi-dimensional binary search. Then, an STL formula is found (valuation of the PSTL template) for each cluster.

The *batch-learning* techniques above all require input data, that is, signals depicting the satisfaction or the violation of properties or behaviour of an autonomous system. All these techniques, however, do not account for a setting where no input data is *a priori* available. Moreover, none of them consider the problem of noisy data, in the sense of adversarial examples. Despite everything, the approach in [5] catches our attention: it consists of a decision tree-based online learning method, incrementally building a candidate specification. In other words, as soon as more signals are available, the candidate specification is updated to take the new observations into account.

III. Definitions

In the following, we reuse most of the notations and definitions of [4]. Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ respectively be the set of reals, positive reals and natural numbers. We use $|r|$ to denote the absolute value of $r \in \mathbb{R}$. Given $x \in \mathbb{R}^n$, its infinity-norm is $\|x\|_\infty := \max_{i\in\{1,\cdots,n\}} |x^i|$, where x^i is the i'th component of x. We use discrete notion of time throughout this paper, and time intervals are in the form $I = [t_1, t_2] \subset \mathbb{N}$, $t_1, t_2 \in \mathbb{N}, t_1 \leq t_2$. $[\tau + t_1, \tau + t_2]$ is denoted by $\tau + I$, $\tau \in \mathbb{N}$. The bounded continuous interval $\{r \mid lb \leq r \leq ub\}$ is denoted by $\mathbb{U}_{[lb,ub]} \subset \mathbb{R}$ with $lb, ub \in \mathbb{R}$. An n-dimensional, real, infinite-time, discrete-time signal s is defined as a string of real values $s : s_0 s_1 s_2 \cdots$, where $s_t \in \mathbb{S}, \mathbb{S} \subset \mathbb{R}^n$, $t \in \mathbb{N}$. We consider $\mathbb{S} = \mathbb{U}^n_{[lb,ub]}$. The set of all signals with values taken in $\mathbb{S}$ is denoted by $\mathcal{S}$. The set of all signal prefixes with time bound T is defined as $\mathcal{S}_T := \{s[0 : T] \mid s \in \mathcal{S}\}$. For the convenience of notation, we use $s \in \mathcal{S}_T$ to say $s[0 : T] \in \mathcal{S}_T$.

A. Signal Temporal Logic

The syntax of STL is defined as follows [3]:

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2 \tag{1}$$

where $\top$ is the Boolean *True* constant, π a predicate over $\mathbb{R}^n$ in the form of $f(x) \sim \mu$, $f : \mathbb{S} \to \mathbb{R}$, $\mu \in \mathbb{U}_{[lb,ub]}$, and $\sim \in \{\leq, \geq\}$; $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction, respectively; and $\mathcal{U}_I$ is the temporal operator *until* over bounded interval I. Other Boolean operations are defined using the conjunction and negation operators to enable the full expression of propositional logic. Additional temporal operators *eventually* and *globally* are defined as

$\Diamond_I \phi \equiv \top \mathcal{U}_I \phi$ and $\Box_I \phi \equiv \neg \Diamond_I \neg \phi$, respectively, where I is an interval. The set of all STL formulae over signals in $\mathcal{S}$ is denoted by $\Phi^{\mathcal{S}}$. The *robustness* of an STL formula is a function $\rho : \mathcal{S} \times \Phi^{\mathcal{S}} \times \mathbb{N} \to \mathbb{R}$, which is recursively defined as [3]:

$$\begin{aligned}
\rho(s,(f(s) \sim \mu),t) &= \begin{cases} \mu - f(s_t) & \sim = \leq \\ f(s_t) - \mu & \sim = \geq \end{cases}, \\
\rho(s,\neg\phi,t) &= -\rho(s,\phi,t), \\
\rho(s,\phi_1 \vee \phi_2,t) &= \max(\rho(s,\phi_1,t),\rho(s,\phi_2,t)), \\
\rho(s,\phi_1 \wedge \phi_2,t) &= \min(\rho(s,\phi_1,t),\rho(s,\phi_2,t)), \\
\rho(s,\phi_1\, \mathcal{U}_I\ \phi_2,t) &= \max_{t' \in t+I} \big(\rho(s,\phi_2,t'), \\
&\qquad \min_{t'' \in [t,t']} \rho(s,\phi_1,t'')\big), \\
\rho(s,\Diamond_I\ \phi,t) &= \max_{t' \in t+I}\ \rho(s,\phi,t'), \\
\rho(s,\Box_I\ \phi,t) &= \min_{t' \in t+I}\ \rho(s,\phi,t').
\end{aligned} \tag{2}$$

A signal *satisfies* an STL specification at a certain time t if and only if its corresponding robustness is positive:

$$s[t] \models \phi \Leftrightarrow \rho(s,\phi,t) \geq 0 \tag{3}$$

The *language* of an STL formula $\phi \in \Phi^{\mathcal{S}}$ is defined as:

$$\mathcal{L}(\phi) = \{s \in \mathcal{S} \mid \rho(s,\phi,0) \geq 0\}\,. \tag{4}$$

The horizon of an STL formula is defined as the minimum length of the time window required to compute its robustness, and it is recursively computed as [20]:

$$\begin{aligned}
&\|\pi\| = 0, \|\phi\| = \|\neg\phi\| \\
&\|\phi_1 \wedge \phi_2\| = \|\phi_1 \vee \phi_2\| = \max\{\|\phi_1\|, \|\phi_2\|\} \\
&\left\|\phi_1 \mathcal{U}_{[t_1,t_2]} \phi_2\right\| = t_2 + \max\{\|\phi_1\|, \|\phi_2\|\} \\
&\left\|\Diamond_{[t_1,t_2]}\phi\right\| = \left\|\Box_{[t_1,t_2]}\phi\right\| = t_2 + \|\phi\|
\end{aligned} \tag{5}$$

The set of all STL formulae over signals in $\mathcal{S}$ such that their horizons are less than T is denoted by $\Phi^{\mathcal{S}_T}$. Note that computing $\rho(s,\phi,t)$ requires $s[t : t + \|\phi\|]$, and the rest of the values are irrelevant.

B. Distance between STL formulae

The (symmetric difference) distance $d : \Phi^{\mathcal{S}} \times \Phi^{\mathcal{S}} \to \mathbb{R}_{\geq 0}$ between two STL formulae ϕ_1 and ϕ_2 is defined as [4, Definition 4]:

$$d(\phi_1,\phi_2) = \frac{1}{\max\{\|\phi_1\|, \|\phi_2\|\} + 1} |\mathcal{L}(\phi_1) \triangle \mathcal{L}(\phi_2)|, \tag{6}$$

where $|\cdot|$ is the Lebesgue measure.

In practice, d can be calculated through the coverage of signal sets in the space-time value set. This is done by computing the symmetric difference of sets of boxes, representing the area of satisfaction of the two STL formulae.

If the distance between two STL formulae is 0, then the subsequent holds (follows from [4, Theorem 2]):

$$\mathcal{L}(\phi_1) = \mathcal{L}(\phi_2) \Leftrightarrow \phi_h \equiv \phi_t \Leftrightarrow d(\phi_h,\phi_t) = 0 \tag{7}$$

C. PSTL Primitives

Parametric Signal Temporal Logic (PSTL) [9] is an extension of STL where formulae are parameterized. The time bounds in the time intervals associated with temporal operators and all the constants in the inequality predicates are replaced by parameters (respectively denoted *time* and *space* parameters). Primitives are defined as simple PSTL formulae, containing one predicate and one or two temporal operators. The set of *first-level* primitives are defined as follows [16, Definition 5.1]:

$$\mathcal{P} = \{\Diamond_{[\tau_1,\tau_2]}(f(x) \sim \mu) \text{ or } \Box_{[\tau_1,\tau_2]}(f(x) \sim \mu)\}, \tag{8}$$

where $\tau_1, \tau_2 \in [0,T]$ are the temporal parameters, $\mu \in \mathbb{U}_{[lb,ub]}$ the spatial parameter and $\sim \in \{\leq, \geq\}$.

IV. Problem Formulation

We want to find an STL formula that describes a property or the behaviour of a system. We consider the setting where no data is *a priori* available, but the learning algorithm can interact with the system to generate data.

Problem 1: Given a (discrete-time) SUT, the behaviour of which is described by a target STL formula ϕ_t, iteratively guess an hypothesis formula ϕ_h such that $d(\phi_h,\phi_t) \to 0$.

V. Proposed Approach

We consider the context of specifications of autonomous systems where no data is available beforehand but are gradually retrieved over time. Therefore, most of the classical temporal logic inference techniques for STL cannot be applied as such. Active learning is a technique where a learning algorithm, the *learner*, iteratively queries a *teacher* to obtain information on new data points, or information on his actual hypothesis model of the solution. Typically, the *learner* can pose a range of queries [21]. Active learning has been successfully applied to a broad array of problems [22]. In this work, we decide to apply active learning to infer STL specifications.

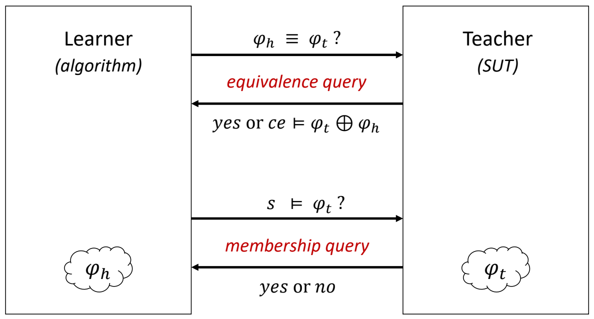


Fig. 1: Active learning of an STL specification. The learning algorithm (*learner*) poses queries (*equivalence* or *membership*) to the SUT (*teacher*).

As depicted in Fig. 1, the learning algorithm (i.e. the *learner*) iteratively builds a *hypothesis specification* ϕ_h by posing *queries* to a *teacher* knowing the target specification ϕ_t of the SUT. The queries can be of several types:

- *membership* queries: the *learner* asks the *teacher* whether a given signal s satisfies the target specification ϕ_t, i.e. $s \models \phi_t$. The teacher returns *yes* or *no* according to its knowledge of ϕ_t. The signal s is generated using any control synthesis method taking as input an STL specification an returning a control strategy complying with the input specification. This can be done, for instance, via sampling-based control synthesis methods including satisfaction of ϕ_h [23] or solving an MILP optimization problem [24].
- *equivalence* queries: the *learner* asks the *teacher* whether the hypothesis specification ϕ_h is equivalent to the target specification ϕ_t, i.e. $\phi_h \equiv \phi_t$. The teacher returns *yes* if $\phi_h \equiv \phi_t$, and the algorithm stops; otherwise it returns *no* with a counterexample signal ce that satisfies the hypothesis specification but not the target specification, or vice versa, i.e. $ce \models \phi_t \oplus \phi_h$. Note that, in practice, equivalence checking is a tricky task. However, in the case of discrete-time systems and sampled time STL, equivalence is decidable using model checking [6]. Indeed, equivalence can be deduced by model checking ϕ_h against the SUT (which answers whether ϕ_h is a good representation of the SUT), as well as $\neg\phi_h$ against the SUT (which gives insight on whether ϕ_h is an over-generalization of the SUT's behaviour). In the rest of the paper, we consider the theoretical case where equivalence can be directly computed through the distance between ϕ_h and ϕ_t.

In the context of monitoring our experiments, we also use *distance* queries: the *learner* asks the *teacher* whether the hypothesis specification ϕ_h is close to the target specification ϕ_t, i.e. asks for $d(\phi_h, \phi_t)$. This is used on two occasions: 1) as termination criterion, i.e. to measure whether the distance between the hypothesis specification and target specification is small enough to consider that ϕ_h converged towards ϕ_t, and 2) to monitor the convergence of ϕ_h towards ϕ_t, and in case the *learner* makes a regression (i.e. an update of the hypothesis leading to an increase of the distance between iterations), help to decide to rollback such an update.

We rely on the online learning algorithm of [5] for the building of the specification. We use metrics on STL [4] to evaluate the quality of our queries.

```
Algorithm 1: ActiveLearnSTL($\phi_t$)
Input: $\alpha$ – convergence factor
    $\beta$ – probability of membership query
    $lb, ub \in \mathbb{R}$ – spatial lower- and upper-bounds
    $\mathcal{P}$ – set of PSTL primitives
Output: $\phi_h$ – hypothesis STL formula
1  $\phi_h \leftarrow \top$
2  $tree, S_-, S_+ \leftarrow initializeDT(\|\phi_t\|, \mathcal{P})$
3  while $d(\phi_h, \phi_t) > \alpha$ do
4      if $Unif([0,1]) > \beta$ then
           // membership query
5          $s \leftarrow generateSignal(\phi_h, lb, ub, \|\phi_t\|)$
6          if $s \models \phi_t$ then
7              $S_- \leftarrow S_- \cup \{s\}$
8              $\phi_h, tree = update(tree, s, S_-)$
9          else
10             $S_+ \leftarrow S_+ \cup \{s\}$
11             $\phi_h, tree = update(tree, s, S_+)$
12     else
           // equivalence query
13         if $\phi_h \not\equiv \phi_t$ then
14             $ce \leftarrow generateSignal(\phi_h \oplus \phi_t, lb, ub, \|\phi_t\|)$
15             if $ce \models \phi_t$ then
16                 $S_- \leftarrow S_- \cup \{ce\}$
17                 $\phi_h, tree = update(tree, ce, S_-)$
18             else
19                 $S_+ \leftarrow S_+ \cup \{ce\}$
20                 $\phi_h, tree = update(tree, ce, S_+)$
21 return $\phi_h$
```

A. Initialization.

Our learning algorithm, as presented in Algorithm 1, takes as input parameters a convergence factor α below which we consider the distance between the hypothesis specification and target specification $d(\phi_h, \phi_t)$ good enough to consider that ϕ_h converged towards ϕ_t. This also serves as a termination criterion of the algorithm, since as soon as $d(\phi_h, \phi_t) \leq \alpha$ holds at any iteration, then the hypothesis specification is returned. The learning algorithm also takes a probability β of triggering either a membership query or an equivalence query at any iteration. Lower- and upper-bounds (respectively lb and ub) for signal generation are set so that the generated signals is defined over $\mathbb{S} = \mathbb{U}^n_{[lb,ub]}$. The algorithm further takes a set $\mathcal{P}$ of PSTL primitives as well as uses information on the time horizon of the target specification $\|\phi_t\|$. The algorithm starts with the initialization of the hypothesis formula to the *True* Boolean constant (line 1), that is, the hypothesis accepts all signals over $\mathcal{S}_{\|\phi_t\|}$. Then, a decision-tree $tree$ is initialized with a single terminal node representing the *True* boolean constant, together with $S_-, S_+ = \emptyset$ respectively denoting the negative and positive dataset (line 2). The algorithm starts next iterating while the hypothesis has not converged towards the target specification (line 3), by either performing membership queries (line 4) or equivalence queries (line 12), where $Unif(X)$ denotes the uniform distribution over set X.

B. Membership Queries.

In the case of membership queries, the process is a follows. First, a signal s is generated based on the knowledge of ϕ_h at any given iteration (line 5). Among possible signal generation methods satisfying an STL specification, we decided to use the boolean encoding of STL in a set of MILP constraints [24, Sect. IV-C]. We omit the encoding details here. Note that, depending on the needs, the same authors propose a quantitative encoding, where signals are generated so that the robustness of the STL formula is maximized. To generate a signal s, parameters lb, ub and $\|\phi_t\|$ are required to ensure that $s \in \mathcal{S}_{\|\phi_t\|}$ with $\mathbb{S} \subset \mathbb{U}^n_{[lb,ub]}$. Once the signal is generated, it is evaluated against the SUT. Therefore, we test whether $s \models \phi_t$ (line 6) which in practice is done by calculating its robustness (Eq. 3), to assign it either to the negative dataset S_- (line 7) or the positive dataset S_+ (line 10). The decision tree $tree$ is then updated according to [5, Algorithm 1].

The way the update is done is as follows: first, given a signal, the decision tree is parsed to retrieve the leaf classifying the signal. The label associated with a leaf is simply chosen using the majority vote on the labels of the signals falling in that leaf. If the signal is misclassified, then an update has to be triggered. The update finds the optimal parameters τ_1, τ_2 and μ for each primitive in the set $\mathcal{P}$ according to misclassification gain [16], [25], and using a nonlinear solver. Second, it evaluates the status of the leaf to decide if it should be kept as a leaf or if a new non-terminal node can be created in its place, and if so, transforms the leaf into a non-terminal node associated with the best parameterized primitive. Two empty leaves (respectively standing for the Boolean *True* and *False* predicates) are initially added as children of this new node. Finally, the signals are partitioned and split to the leaves according to the formula found.

We also keep ϕ_h updated by translating the decision tree into the hypothesis specification, according to [16, Algorithm 2]. ϕ_h is then obtained by recursively parsing the tree's nodes. To do so, at each node, we calculate: 1) the conjunction of the node's formula with the left subtree's formula, 2) the conjunction of the negation of the node's formula with its right subtree's formula, and finally return the disjunction of 1 and 2.

C. Equivalence Queries.

In the case of equivalence queries, equivalence of ϕ_h and ϕ_t is first tested (line 13). In other words, since $\phi_h \equiv \phi_t \Leftrightarrow d(\phi_h, \phi_t) = 0$ (Eq. 7), we do this by computing the distance between the two STL formulae. In case the two formulae are not equivalent, we then generate a counterexample $ce \in \mathcal{S}_{\|\phi_t\|}$ s.t. $ce \models \phi_h \oplus \phi_t$, that is, a signal in the symmetric difference of the languages of the two STL formulae (line 14). The counterexample is then evaluated against ϕ_t and added to the corresponding dataset. The decision tree and the hypothesis specification are updated accordingly (lines 16 – 20) as described in Section V-B.

Note here that for the generation of the counterexample, we use the symmetric difference of the hypothesis and target specifications, instead of a counterexample $ce \in \mathcal{L}(\phi_t)$. Even though the initialization of ϕ_h is set to $\top$, and therefore $\phi_h \supseteq \phi_t$, it may be the case that some over-specializations may occur during the iterative process. As a result, there may exist signals $s \in \mathcal{L}(\phi_t), s \notin \mathcal{L}(\phi_h)$, and $\phi_h \not\supseteq \phi_t$. The inclusion of ϕ_t into ϕ_h is a desirable property to aim for convergence of ϕ_h towards ϕ_t, therefore an equivalence query where a counterexample $ce \in \mathcal{L}(\phi_h) \triangle \mathcal{L}(\phi_t)$ can help recovering from such over-specialization not covering the entire semantics of ϕ_t.

D. Properties.

In this section, we conjecture some properties of our active learning algorithm. Proofs are outlined for the sake of brevity.

Proposition 5.1: Given Algorithm 1, the horizon of the hypothesis specification $\|\phi_h\|$ converges towards the horizon of the target specification $\|\phi_t\|$.

Note that by construction of the signals, these are defined over $\|\phi_t\|$. By construction of the $update$ function, rules will be added incrementally with maximum horizon $\|\phi_t\|$. We can, therefore, expect a convergence of the horizon of ϕ_h towards the horizon of ϕ_t, the more new signals are available and the hypothesis specification updated.

Proposition 5.2: Given Algorithm 1, the hypothesis specification ϕ_h converges towards target specification ϕ_t.

Partial correctness of Algorithm 1 is trivial since learning only terminates after the equivalence of the inferred model is guaranteed. Termination is, however, more complicated to prove. Some properties, such as the equivalence checking mechanism, indicate termination. Indeed, equivalence checking provides new counterexamples as long as the hypothesis STL formula does not match the desired target STL formula. The definition of canonical hypothesis STL formula given a set of signals could help to prove termination. Working under tightness is among theoretical developments we leave for future work.

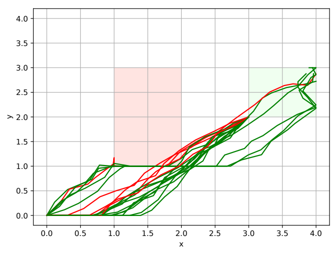
VI. Experiments

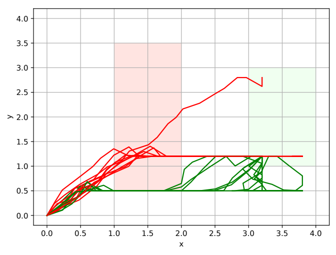
We implemented and tested our active learning algorithm in Python 3.8[1], using SciPy-particle swarm libraries for the optimization of PSTL primitives, and the Gurobi optimizer [26] to solve the MILP problem for signal generation. We ran our experiments on an Intel i7-8665U CPU and 32GB RAM. We consider a set of motion planning scenarios in a 2-dimensional space. We apply our technique to 4 examples, whose spatial representations are shown in Figure 2:

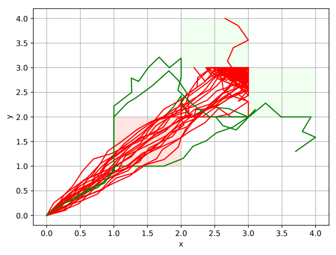
$$\begin{aligned}
\phi_1 &= \square_{[25,30]}(x > 3 \land x < 4 \land y > 2 \land y < 3) \\
&\land \square_{[0,30]} \neg(x > 1 \land x < 2 \land y > 1 \land y < 3) \\
\phi_2 &= \lozenge_{[0,30]}(x > 3 \land x < 4 \land y > 1 \land y < 3) \\
&\land \square_{[0,30]}\neg(x > 1 \land x < 2 \land y > 0.5 \land y < 3.5) \\
\phi_3 &= (\lozenge_{[0,30]}(x > 2 \land x < 3 \land y > 3 \land y < 4) \\
&\lor \lozenge_{[0,30]}(x > 3 \land x < 4 \land y > 2 \land y < 3)) \\
&\land \square_{[0,30]}\neg(x > 1 \land x < 2 \land y > 1 \land y < 2) \\
\phi_4 &= \lozenge_{[10,20]}(x > 0 \land x < 1 \land y > 3 \land y < 4) \\
&\land \lozenge_{[30,50]}(x > 3 \land x < 4 \land y > 0 \land y < 1) \\
&\land \square_{[0,50]}\neg(x > 2 \land x < 3 \land y > 1 \land y < 2)
\end{aligned}$$

In each of the scenarios, the initial coordinates are (0,0) and the lower- and upper-bounds set to 0 and 4, respectively. Moreover, we divide the experiments into three parts: the first one tries to guess the entire STL formula. In the second experiment, the *mission* (green) is already given prior to the initialization of the algorithm, the *spatial preference* (red) being left to learn. Finally, the third experiment is done with

[1]Software available at https://github.com/allinard/active-learn-stl


(a) ϕ_1


(b) ϕ_2


(c) ϕ_3

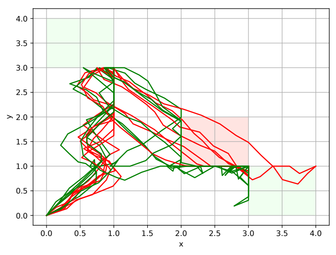

(d) ϕ_4

Fig. 2: Spatial representations of 4 different STL formulae, with satisfying (green) and violating (red) signals.

a different set of primitives, directly encapsulating desired templates of STL formulae fitting better our motion planning scenarios. For all our experiments, we use a basic control policy integrated into the signal generation, such that $\forall t \in [0, T], t \in \mathbb{N}, |s[t] - s[t+1]| < 0.2 \pm \epsilon$ with $\epsilon \in [0, 0.05]$.

A. *Learning mission and spatial preference.*

	time (s)	#iterations	distance	$\|\phi_h\|$
ϕ_1	7652	901	0.75568	13
ϕ_2	3303	224	0.63288	11
ϕ_3	2235	161	0.31270	4
ϕ_4	16650	871	0.42125	2

TABLE I: Results for learning the whole STL formulae.

In this experiment, the goal is to learn the entire STL target specifications $\phi_1 \dots \phi_4$. In Table I, we present the results we achieve for each target specification. The maximum number of iterations is set to 1000. We show the results of the best hypothesis specification found in terms of distance, and show at which iteration it is found, runtime, and also the size $|\phi_h|$ of the candidate STL formula in terms of the number of primitives it contains.

B. *Learning spatial preference only.*

	time (s)	#iterations	distance	$\|\phi_h\|$
ϕ_1	4251	835	0.48856	9
ϕ_2	27016	659	0.17392	7
ϕ_3	12014	1000	0.37396	2
ϕ_4	3924	260	0.41768	11

TABLE II: Results for learning *spatial preferences* only (*mission* given as input information).

Here, we consider a setting where the *mission* is already known, but the *spatial preference* remains to be inferred. We present the results achieved in Table II. As in the previous experiment, the maximum number of iterations is set to 1000. However, the hypothesis specification is initialized with the mission formulae, as well as the decision tree, initialized with a non-terminal node containing the mission formulae, a leaf for the negative labels, and an invariant leaf for the classification of the positive signals.

	time (s)	#iterations	distance	$\|\phi_h\|$
ϕ_1	24792	341	0.32742	10
ϕ_2	23688	640	0.20041	15
ϕ_3	59660	737	0.35571	16
ϕ_4	21570	781	0.41582	16

TABLE III: Results for learning the whole STL formulae, with spatial and mission primitives.

C. *Learning with spatial and mission primitives.*

Finally, in this section, we consider the case of learning with a set of primitives which fit better the needs of our experiments. Given the specific scenario we consider (a motion planning task with spatial preferences), we run our algorithm with a set of spatial and motion primitives defined as follows:

$$\mathcal{P}_{spatial} = \{\square_{[\tau_1,\tau_2]}\neg(x \geq \mu_1 \wedge x \leq \mu_2 \wedge y \geq \mu_3 \wedge y \leq \mu_4)\}$$

$$\mathcal{P}_{mission1} = \{\diamondsuit_{[\tau_1,\tau_2]}(x \geq \mu_1 \wedge x \leq \mu_2 \wedge y \geq \mu_3 \wedge y \leq \mu_4)\}$$

$$\mathcal{P}_{mission2} = \{\square_{[\tau_1,\tau_2]}(x \geq \mu_1 \wedge x \leq \mu_2 \wedge y \geq \mu_3 \wedge y \leq \mu_4)\}$$

The goal of the *update* function called in Algorithm 1 is to find the optimal parameters τ_1, τ_2, μ_1, μ_2, μ_3, μ_4 of the 3 primitives $\mathcal{P}_{spatial}$, $\mathcal{P}_{mission1}$, and $\mathcal{P}_{mission2}$. The results are presented in Table III.

D. *Discussion.*

We can see that good results can be obtained whenever some knowledge on the SUT is incorporated into the learning algorithm. Our best results are achieved when using primitives strongly fitting the specification to guess. The definition of PSTL primitives matching expectations of the goal specification is not unrealistic. In practice, we argue that the nature of the problem or believes in the behaviour of the system are already known, and represent reasonable assumptions on the SUT.

Our approach contains several limitations. The first one is the need for a proper balancing of the signals. Indeed, by construction of the decision tree, an unbalanced dataset poorly leads to proper separation of the observed signals. This effect can be counterbalanced by fine-tuning of the parameter α, since positive signals tend more to be obtained through equivalence queries than thanks to membership queries. This is even more obvious after the few iterations after initialization, since one can have low hopes to generate a positive signal when the hypothesis specification is still the *True* boolean constant. Indeed, such signal can potentially be any signal in $\mathcal{S}$. An additional limitation is also the need for information on the horizon of ϕ_t to generate signals of proper length. We argue that this is also a reasonable assumption on the SUT.

A research question raised by our method also lays on the definition of a *Minimally Adequate Teacher*. In our scenario, we consider that the teacher can answer membership and equivalence queries. In practice, the answering of membership queries is trivial. However, answering an equivalence query is more sophisticated. It requires either strong assumptions on the system to learn or precise knowledge of its expected behaviour. We leave these theoretical questions, as well as real-world applications for future work.

VII. Conclusion and Future Work

Our technique enables the inference of an STL specification of a system on which no data is *a priori* available. We consider the particular case of active learning, where a system can answer queries on the membership of its behaviour, or the equivalence of a hypothesis specification to its specification. However, once put in practice, one drawback of our approach as such is intrinsic to the quality of the answers to the queries given by the *teacher*. In a practical context, the answer to the queries can be done thanks to an oracle that would reproduce the target behaviour of the SUT.

In the future, we will investigate extensions of our approach, for instance, the possibility of *correction* queries (given a violating signal, ask the teacher to show which parts of the signal violate the target specification and should be modified to satisfy the specification). We also plan to use non-invasive techniques [27] to preserve the safety requirements inherent to the system. They ensure the system to be correct against a given safety specification, that should be known beforehand. In other words, they ensure that the actions were taken by the *learner* fulfil safety requirements, and correct them if necessary. We also envision a wide range of application for our technique, especially in the domain of Human-Robot Interaction, and especially to mine social requirements of autonomous systems. To that end, we plan to use crowdsourcing as oracle interfacing the learning algorithm and the target specification.

References

[1] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2005, pp. 2020–2025.

[2] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1988–1993.

[3] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[4] C. Madsen, P. Vaidyanathan, S. Sadraddini, C.-I. Vasile, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Metrics for signal temporal logic formulae," in *Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 1542–1547.

[5] G. Bombara and C. Belta, "Online learning of temporal logic formulae for signal classification," in *European Control Conference (ECC)*. IEEE, 2018, pp. 2057–2062.

[6] H. Roehm, J. Oehlerking, T. Heinz, and M. Althoff, "STL model checking of continuous and hybrid systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2016, pp. 412–427.

[7] D. Neider and I. Gavran, "Learning linear temporal properties," in *Formal Methods in Computer Aided Design*. IEEE, 2018, pp. 1–10.

[8] R. Yan, Z. Xu, and A. Julius, "Swarm signal temporal logic inference for swarm behavior analysis," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3021–3028, 2019.

[9] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *International Conference on Runtime Verification*. Springer, 2011, pp. 147–160.

[10] A. Bakhirkin, T. Ferrère, and O. Maler, "Efficient parametric identification for STL," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*. ACM, 2018, pp. 177–186.

[11] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "TeLEx: Passive STL learning using only positive examples," in *International Conference on Runtime Verification*. Springer, 2017, pp. 208–224.

[12] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.

[13] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proceedings of the 17th International conference on Hybrid systems: computation and control (HSCC)*. ACM, 2014, pp. 273–282.

[14] Z. Kong, A. Jones, and C. Belta, "Temporal logics for learning and detection of anomalous behavior," *Transactions on Automatic Control*, vol. 62, no. 3, pp. 1210–1222, 2016.

[15] L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi, "A robust genetic algorithm for learning temporal specifications from data," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2018, pp. 323–338.

[16] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 2016, pp. 1–10.

[17] A. Jones, Z. Kong, and C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach," in *53rd Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 848–853.

[18] P. Vaidyanathan, R. Ivison, G. Bombara, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Grid-based temporal logic inference," in *56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 5354–5359.

[19] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 305–325.

[20] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *International Conference on Runtime Verification*. Springer, 2014, pp. 231–246.

[21] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, no. 4, pp. 319–342, 1988.

[22] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[23] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3840–3847.

[24] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 81–87.

[25] B. D. Ripley and N. Hjort, "Pattern recognition and neural networks," 1995.

[26] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[27] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 2669–2678.