Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Learning Interpretable Models Expressed in Linear Temporal Logic

Alberto Camacho, Sheila A. McIlraith
Department of Computer Science, University of Toronto
Vector Institute
Toronto, Canada
{acamacho, sheila}@cs.toronto.edu

Abstract

We examine the problem of learning models that characterize the high-level behavior of a system based on observation traces. Our aim is to develop models that are human interpretable. To this end, we introduce the problem of learning a *Linear Temporal Logic* (LTL) formula that parsimoniously captures a given set of positive and negative example traces. Our approach to learning LTL exploits a symbolic state representation, searching through a space of labeled skeleton formulae to construct an alternating automaton that models observed behavior, from which the LTL can be read off. Construction of interpretable behavior models is central to a diversity of applications related to planning and plan recognition. We showcase the relevance and significance of our work in the context of behavior description and discrimination: i) active learning of a human-interpretable behavior model that describes observed examples obtained by interaction with an oracle; ii) passive learning of a classifier that discriminates individual agents, based on the human-interpretable signature way in which they perform particular tasks. Experiments demonstrate the effectiveness of our symbolic model learning approach in providing human-interpretable models and classifiers from reduced example sets.

1 Introduction

Constructing a model of system behavior from observation traces, in a form that is understandable and meaningful to a human, is central to human interpretability of complex systems. Model learning can be used to learn temporally extended (i.e., non-Markovian) patterns such as safety and reachability rules, and other complex behaviors, and it can be composed with other techniques such as imitation learning and inverse reinforcement learning of reward functions. Acquired models can be exploited for verification of system properties, plan recognition, behavior discrimination, and for knowledge extraction and transfer.

We are motivated to learn human-interpretable models from observation traces with a view to addressing two interesting problems: (1) discriminatory and explainable plan recognition – recognizing an agents plan from among a finite set of options and explaining why a particular classification was made relative to other options under consideration; and (2) the related problem of recognizing an individual based upon the way in which they perform a particular task, as compared to others, and recounting the behavioral signature that makes them unique.

Central to both of these problems, and to a diversity of other problems, is the notion of learning a human interpretable model from observation traces. As such the bulk of this paper focuses on the challenging task of model learning, and we return to specific application tasks in the last section of the paper where we evaluate our model learning method.

The objective in model learning is to construct a model that is consistent with a set of positive and negative examples. In this paper, examples are *finite* state traces. There exists a whole body of work on learning regular languages, starting with the seminal work by Angluin (1987) on L^* – an algorithm that learns finite-state machines and, in particular, deterministic finite-state automata (DFA). Further work on learning other types of finite-state machines and automata has been conducted (e.g., (Bollig et al. 2009; Giantamidis and Tripakis 2016; Angluin, Eisenstat, and Fisman 2015; Smetsers, Fiterau-Brostean, and Vaandrager 2018)). More recently, there has been work on software program *specification mining* to capture temporal properties (e.g., (Gabel and Su 2008; 2010)) using automata representations of regular expressions.

We are interested in learning human-interpretable models. Unfortunately, the number of states and state transitions in automata representations of regular languages is often too large and complex to be understood by a human. We wish to learn models in a high-level language that facilitates human interpretation and manipulation. To this end, we introduce the problem of model learning of a *Linear Temporal Logic* (LTL) formula. LTL has a natural syntax that many find compelling. It was originally developed to specify properties of programs for verification (Pnueli 1977), and has subsequently been used to specify properties for automated synthesis of reactive systems, also known as LTL synthesis (Pnueli and Rosner 1989).

Learning LTL patterns from log data has also been investigated in several software engineering contexts, largely to discover temporal rules or to search for specific sequencing invariant patterns in software systems (e.g., (Lemieux, Park, and Beschastnikh 2015)). In the context of AI planning, de la Rosa and McIlraith (2011) learn LTL models

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

with a restricted syntax. These models were used as domain control knowledge to guide search in TLPlan (cf. (Bacchus and Kabanza 2000)). Most recently Kasenberg and Scheutz (2017) addressed the problem of LTL pattern mining of observation traces in service of interpretable apprenticeship learning in MDPs. Their work shares high-level motivation with our work, but differs with respect to how it is realized, formulating the task as a multi-objective optimization problem and employing genetic programming to solve it. Also related to our work is the work by Neider and Gavran (2018), which learns LTL temporal properties (interpreted over infinite traces) that are consistent with a given set of finite observation traces.

In this paper we use LTL interpreted over *finite* traces, commonly referred to as LTL_f, to characterize families of observation traces. LTL_f has been used in AI planning to specify temporally extended goals (e.g. (Bacchus and Kabanza 2000; Baier and McIlraith 2006; Camacho et al. 2017b)) and preferences (e.g., (Edelkamp 2006; Baier, Bacchus, and McIlraith 2009; Coles and Coles 2011; Bienvenu, Fritz, and McIlraith 2011)). Likewise, LTL_f serves as specification language for so-called LTL_f synthesis (e.g. (De Giacomo and Vardi 2015; Zhu et al. 2017; Camacho et al. 2018) and LTL_f synthesis with environment assumptions (Camacho, Bienvenu, and McIlraith 2018). Learning LTL_f models has immediate applications to learning specifications for LTL_f synthesis, planning and inverse reinforcement learning. We revisit other potential applications in Section 7.

We study two learning paradigms: *passive* learning of LTL_f, where the objective is to learn a formula that is consistent with a given set of positive and negative examples; and *active* learning of LTL_f, where the objective is to learn a formula by interacting with an oracle. We identify potential uses of LTL_f model learning, and evaluate the advantages of our methods relative to alternative approaches.

In addition to producing a highly interpretable solution, we show that LTL_f learning can be conducted with exponentially fewer examples than is required to learn DFA, both theoretically and in practice. This property positions our system well for few-shot learning – i.e., learning models that generalize from a reduced set of examples.

Finally, we conducted experiments in behavior classification, where passive LTL_f model learning can be used to obtain behavior classifiers. We obtained an accuracy that is comparable to state-of-the-art deep learning approaches to time series classification. Our system learned orders of magnitude faster, and produced interpretable models.

2 Preliminaries: Linear Temporal Logic

LTL is a modal logic typically used to express temporally extended constraints over state trajectories (Pnueli 1977). The syntax of LTL for a finite set of propositions $p \in \mathrm{AP}$ includes the standard logic connectives ($\wedge, \vee, \neg$), true and false symbols, and temporal operators *next* (X) and *until* (U).

$$\varphi := \mathsf{false} \mid \mathsf{true} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathsf{X}\,\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

Operators *release* (R), *eventually* (F), and *always* (G) are commonly used, and can be defined as: $\varphi_1 \,\mathsf{R}\, \varphi_2 := \neg(\neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2)$, $\mathsf{F}\,\varphi := \mathsf{true}\,\mathsf{U}\,\varphi$, and $\mathsf{G}\,\varphi := \neg(\mathsf{true}\,\mathsf{U}\,\neg\varphi)$. Typically, the truth of an LTL formula φ is interpreted over *infinite* traces of propositional states – that is, sequences of elements in 2^{AP}. An infinite trace $w = \sigma_0\sigma_1 \cdots$ *satisfies* φ when $w, 0 \models \varphi$, also written as $w \models \varphi$. Intuitively, $w, i \models \psi$ denotes that the suffix of w from σ_i satisfies ψ. Formally:

$$w, i \models p \text{ iff } p \in AP \text{ and } \sigma_i \models p$$
$$w, i \models \neg\alpha \text{ iff } w, i \not\models \alpha$$
$$w, i \models \alpha \wedge \beta \text{ iff } w, i \models \alpha \text{ and } w, i \models \beta$$
$$w, i \models \mathsf{X}\,\alpha \text{ iff } w, i+1 \models \alpha$$
$$w, i \models \alpha \,\mathsf{U}\, \beta \text{ iff } w, k \models \beta \text{ for some } i \leq k < \infty \text{ and } w, j \models \alpha \text{ for all } i \leq j < k$$

Different fragments and semantics of LTL have been studied. In this paper we are concerned with LTL interpreted over *finite* traces $w = \sigma_0 \cdots \sigma_{n-1}$ (e.g. (Baier and McIlraith 2006; De Giacomo and Vardi 2013)), also refered to as *finite* LTL, and more recently as LTL_f. The semantics of an LTL_f formula φ is similar to the semantics of LTL, but differs in the interpretations of the X and U operators as shown below:

$$w, i \models \mathsf{X}\,\alpha \text{ iff } i+1 < n \text{ and } w, i+1 \models \alpha$$
$$w, i \models \alpha \,\mathsf{U}\, \beta \text{ iff } w, k \models \beta \text{ for some } i \leq k < n \text{ and } w, j \models \alpha \text{ for all } i \leq j < k$$

Notably, the semantics of LTL_f has to account for the end of the trace, detected when $w, i \models \neg\,\mathsf{X}\,\mathsf{true}$. Operator *weak next* (N) defined by $\mathsf{N}\,\alpha \equiv \neg\,\mathsf{X}\,\mathsf{true} \vee \mathsf{X}\,\alpha$, is used to denote that α needs to hold in the next step if it exists (e.g. (De Giacomo and Vardi 2013)). In LTL, operator *weak next* is equivalent to operator *next*, and $\neg\,\mathsf{X}\,\alpha \equiv \mathsf{X}\,\neg\alpha$. In LTL_f, $\neg\,\mathsf{X}\,\alpha \equiv \mathsf{N}\,\neg\alpha$ is not equivalent to $\mathsf{X}\,\neg\alpha$.

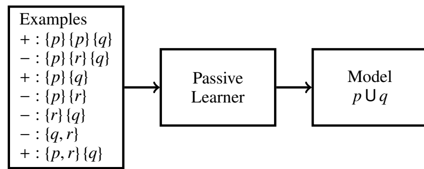
Definition 1. *The size of an LTL_f formula φ is the number of temporal operators, logical connectives, and literals in φ.*

In this paper, we assume LTL_f formulae are expressed in *negation normal form* (NNF), unless otherwise noted. The NNF of an LTL_f formula φ is a formula φ' equivalent to φ where negations ($\neg$) only appear at the level of propositional formulae. LTL_f formulae can be rewritten in NNF in linear time by pushing negations to the level of propositional formulae. NNF transformations preserve the formula size.

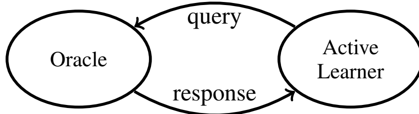
Finally, a premise of this paper is that LTL/LTL_f is more interpretable than automata or other formal languages. Our claim is supported in small measure by the wide use of LTL by non-experts for the specification of user intent in the development of software systems. Indeed, the declarative and compositional nature of LTL shares much with natural languages. Replacing LTL operators with their English equivalents – *next* for X, *eventually* for F, *always* for G, etc. – further illustrates this claim, e.g., *"Always have-key."* or *"Eventually at-home"*.

3 The Problem: Learning Finite LTL Models

We are concerned with the problem of learning a high-level description of the behavior observed in a given set of positive (E^+) examples, while excluding behavior observed in a set of negative (E^-) examples. Having effective means to recognize and model patterns in observed behavior is useful to the design of intelligent agents that need to generalize to previously unseen situations. This includes learning of safety rules and reachability objectives, as well as



(a) Workflow of passive learning



(b) Interaction between the active learner and the oracle.

Figure 1: Illustrative representations of passive and active learners. Figure 1a exemplifies a passive learner that learns an LTL_f formula that is consistent with a given set of positive (+) and negative (−) examples. States are truth assignments to AP := $\{p, q, r\}$, represented as sets $s \subseteq$ AP, and interpreted with the closed-world assumption. In contrast, an active learner (Figure 1b) learns from interaction with an oracle, by performing membership and equivalence queries.

temporally extended formulae that capture complex behavior, like pumping up a bicycle tire. Models can provide domain knowledge, that can be used to reward behaviors in MDPs (Bacchus, Boutilier, and Grove 1996) and to prune the search space in planning (Bacchus and Kabanza 1998).

Related to this problem there is a body of work on *active* and *passive* learning of regular languages, starting from the seminal work by Angluin (1987) on learning deterministic finite-state automata with the L^* algorithm, to more recent work on learning DFA (Giantamidis and Tripakis 2016), non-deterministic automata (Bollig et al. 2009), alternating automata (Angluin, Eisenstat, and Fisman 2015), and finite-state machines (Smetsers, Fiterau-Brostean, and Vaandrager 2018). These approaches use explicit state representations, and suffer from scalability. It has been observed that the number of states in the model hypothesis becomes a bottleneck to model learning, as many learning algorithms are quadratic in this number. The key to scale is reducing the number of states with some sort of abstraction (Vaandrager 2017). LTL_f realizes this by allowing for compact representation of state properties at an abstract level. We adopt LTL_f as our behavior description language.

Passive Learning *Passive learning* is the problem of constructing a model that is consistent with a given set of examples (cf. Figure 1a). Examples can come from a teacher, or an oracle that responds to membership and equivalence queries (c.f. (Angluin 1987)). In this paper, we address the problem of passive model learning of LTL_f formulae (Definition 2). Throughout the paper, we consider examples e which are finite sequences of observations, $e = o_1 \cdots o_n$. An observation o denotes what is *true* in a world state s. We represent observations as subsets $o \subseteq$ AP, where AP is a finite set of propositions. Observations are interpreted with a *closed-world assumption*, as is done in databases and in a number of knowledge representation languages: anything that is not in o is assumed not to hold in a state, s.

Definition 2. Passive model learning of LTL_f *relative to a disjoint set of positive and negative examples involves computing an* LTL_f *formula that entails all positive examples, and none of the negative examples.*

Active Learning An active learner performs queries to an oracle, which knows the target model (cf. Figure 1b). *Membership queries* ask whether an example belongs to the language of the target model. *Equivalence* queries ask whether a guessed model corresponds to the target. When this is not the case, the oracle responds with a counterexample, i.e., a positive example that is not modeled by the guessed model, or a negative example that is modeled by the guessed model.

Definition 3. Active model learning *relative to a target* LTL_f *formula involves computing an* LTL_f *formula logically equivalent to the target by performing membership and equivalence queries to an oracle.*

An active learner can always be constructed from a passive learner that only performs equivalence queries (Walkinshaw, Derrick, and Guo 2009), as illustrated in Algorithm 1. Other forms of active learning can be designed that take into account the cost of performing different queries, available resources, and whether the oracle is an adequate teacher. In what follows, we largely focus on passive learning and leave the development of efficient active learners for future work.

Algorithm 1 Active learning of a minimal LTL_f formula.

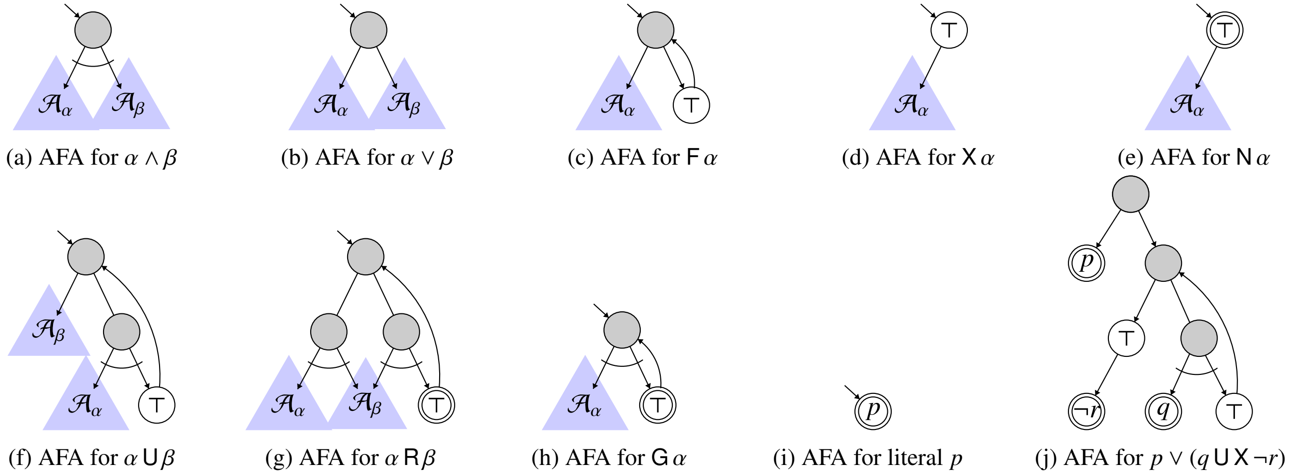
1: Fix $N = 1$, $E = \emptyset$
2: Do passive learning of an LTL_f formula φ of size bounded by N consistent with the examples in E.
3: If no model exists, increment N by one and go to 2.
4: Perform an equivalence query with φ.
5: If φ is not equivalent to the target model, add counterexample to E and go to 2. Else, return φ.

Simpler models are commonly preferred because they tend to generalize better. This *parsimony principle* is typical in model learning, including learning of automata and state machines. In this context, there may be a benefit to learning minimal LTL_f formulae. (i.e., a formula of minimum size).

4 Skeletons for Finite LTL

Many approaches to reasoning with LTL_f exploit the correspondence between LTL_f formulae and finite state automata (e.g. (Baier and McIlraith 2006)). In this work, we exploit their correspondence with *alternating finite automata* (AFA) (Chandra, Kozen, and Stockmeyer 1981).

We first introduce AFA and well-known transformations of LTL_f into AFA (Vardi 1997). Then, we introduce *skeletons* of LTL_f subformulae. Skeletons allow for a modular construction of AFA that preserves the structure of the LTL_f formula. In Section 5 we present a method to learn an LTL_f formula by constructing an AFA, and recovering the formula from its skeleton structure.



(a) AFA for $\alpha \wedge \beta$ (b) AFA for $\alpha \vee \beta$ (c) AFA for $\mathsf{F}\,\alpha$ (d) AFA for $\mathsf{X}\,\alpha$ (e) AFA for $\mathsf{N}\,\alpha$

(f) AFA for $\alpha\,\mathsf{U}\,\beta$ (g) AFA for $\alpha\,\mathsf{R}\,\beta$ (h) AFA for $\mathsf{G}\,\alpha$ (i) AFA for literal p (j) AFA for $p \vee (q\,\mathsf{U}\,\mathsf{X}\,\neg r)$

Figure 2: Figures 2a to 2i: AFA skeletons of LTL and LTL_f subformulae. Nodes are represented by circles. Accepting states are indicated by double circles. Figure 2j: example AFA for $p \vee (q\,\mathsf{U}\,\mathsf{X}\,\neg r)$.

4.1 AFA for LTL on Finite Traces

An AFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is a finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state of the automaton, $\delta : Q \times \Sigma \to \mathrm{B}^+(Q)$ is a transition function that maps pairs (q, s) to positive boolean formulae over $Q \cup \{\top\}$, and $F \subseteq Q$ is a set of accepting states. A *run* of an AFA on a finite word $w = \sigma_0 \cdots \sigma_{n-1}$ is a Q-labeled tree. The root node, at depth 0, is labeled with q_0. The children of a node N at level i labeled with q is a (possibly empty) set of nodes at level $i+1$ with labels q_j such that $\bigwedge_j q_j \models \delta(q, \sigma_i)$. In particular, if node N has no children, then it must be $\delta(q, \sigma_i) = \top$. A run of $\mathcal{A}$ on w is accepting if all nodes at depth n, if any, are labeled by accepting states. Finally, an AFA $\mathcal{A}$ *accepts* a finite word w if some run of $\mathcal{A}$ on w is accepting. Runs in AFA are tree expansions, and differ from the runs of (perhaps more commonly used) finite-state automata which are tree branches.

Definition 4. *The size of an AFA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$ is its number of states, $|Q|$.*

An LTL_f formula φ can be transformed into an AFA $\mathcal{A}_\varphi$ that accepts all, and only the finite traces that satisfy φ. Here we use a construction that follows (De Giacomo, Masellis, and Montali 2014), and is analogous to the progression of LTL_f formula that appeared in (Torres and Baier 2015). The transformation can be done in linear time (Theorem 1). For the moment, we use an auxiliary predicate final that is true in the last state of the trace. As we will observe later on, our learning approach does not require the use of it.

Theorem 1. *An AFA that accepts the models of an LTL_f formula φ can be constructed in linear time in the size of φ.*

The AFA transformation of an NNF LTL_f formula φ is $\mathcal{A}_\varphi := \langle 2^{AP}, sub(\varphi), q_\varphi, \delta, F\rangle$, with components described below. Accepting states are those of the form $\alpha\,\mathsf{R}\,\beta$. The set of states is $sub(\varphi)$, where elements in $sub(\varphi)$ are the subformulae of φ. The initial state is φ, and the set of accepting states are those ψ of the form $\neg(\alpha\,\mathsf{U}\,\beta)$. We use an auxiliary function dual, defined as $\mathsf{dual}(\psi) = \neg\psi$, and $\mathsf{dual}(\neg\psi) = \psi$. The function δ is described below.

$\delta(p, s) := \top$ if $p \in s$ $\quad$ $\delta(p, s) := \bot$ if $p \notin s$

$\delta(\alpha \vee \beta, s) := \delta(\alpha, s) \vee \delta(\beta, s)$ $\quad$ $\delta(\alpha \wedge \beta, s) := \delta(\alpha, s) \wedge \delta(\beta, s)$

$\delta(\neg\alpha, s) := \mathsf{dual}(\delta(\alpha, s))$

$\delta(\alpha\,\mathsf{U}\,\beta, s) := \delta(\beta, s) \vee (\delta(\alpha, s) \wedge \alpha\,\mathsf{U}\,\beta)$

$\delta(\alpha\,\mathsf{R}\,\beta, s) := (\delta(\alpha, s) \wedge \delta(\beta, s)) \vee (\delta(\beta, s) \wedge \delta(\mathsf{N}\,(\alpha\,\mathsf{R}\,\beta)))$

$\delta(\mathsf{N}\,\alpha, s) := \delta(\mathsf{final}, s) \vee \delta(\mathsf{X}\,\alpha, s)$

$$\delta(\mathsf{X}\,\alpha, s) := \begin{cases} \bot, & \text{if } \mathsf{final} \in s \\ \alpha, & \text{if } \mathsf{final} \notin s \end{cases}$$

Property 1. *The size of NNF formula φ equals the size of $\mathcal{A}_\varphi$.*

4.2 AFA Skeletons

We represent AFA transformations of LTL_f formulae by composition of a set of basic structures, that we call *skeletons*. A skeleton is a graph whose root node represents a subformula, state transitions replicate the transition function δ for AFA, and leaf nodes represent other subformulae. Skeleton structures are similar to the AFA representations published in (Finkbeiner and Sipma 2004).

Figure 2 shows skeleton representations of the LTL_f subformulae considered in this paper. Nodes are represented by circles. The root node of a skeleton ξ_φ represents subformula φ. Triangles represent skeletons for the subformulae of φ. Node transitions replicate the transition function δ. Two types of node transitions exist: *universal* ("and") transitions are indicated by edges connected with arcs, as in Figure 2a, that model conjunctive formulae in the definition of δ; *existential* ("or") transitions are indicated by unconnected edges, as in Figure 2b, that model disjunctive formulae in δ.

A tree expansion of a skeleton ξ is defined as follows. Starting from the root node of ξ, each node is expanded by either one of its existential transitions, or by all its universal transitions. Leaf nodes that represent skeletons ξ' are replaced by a tree expansion of ξ'. White circles represent the start of a new level in the tree expansion. White nodes are

labeled with either a literal, or $\top$. A tree expansion is *valid* wrt finite word $w = \sigma_0 \cdots \sigma_{n-1}$ when each σ_i satisfies the labels of all white nodes at level i. A valid tree expansion wrt w is *accepting* if all white nodes at depth n are accepting, indicated by double circles.

Valid tree expansions of an skeleton ξ_φ wrt a finite word w simulate the runs of an AFA $\mathcal{A}_\varphi$ on w. In particular, accepting runs of $\mathcal{A}_\varphi$ have its counterpart in accepting tree expansions of ξ_φ. Note that construction of ξ_φ for an LTL_f formula does not require the use of the auxiliary predicate final to indicate the end of the finite words. The skeleton structures that we use here resemble AFA representations for LTL over finite traces used in (Finkbeiner and Sipma 2004).

Example: Figure 2j represents an AFA for LTL_f formula $\varphi := p \vee (q \, \mathsf{U} \, \mathsf{X} \neg r)$. The AFA is constructed by composing the skeletons of the subformulae of φ, namely: $(p \vee (q \, \mathsf{U} \, \mathsf{X} \neg r)), (p), (q \, \mathsf{U} \, \mathsf{X} \neg r), (q), (\mathsf{X} \neg r)$, and $(\neg r)$. Subformulae that are literals are represented with a skeleton in the form of Figure 2i. Subformulae that contain temporal or logical operators are represented with a skeleton for such operators. The AFA in Figure 2j accepts the finite words that satisfy φ, such as $w = \{\neg p, q, \neg r\}\{p, \neg q, r\}\{p, \neg q, \neg r\}$. This can be seen by expanding the AFA tree that checks q in the first level, checks $\top$ (and commits to $\mathsf{X} \neg r$) in the second level, and checks $\neg r$ in the third level.

5 Passive Learning of Finite LTL

Our approach to passive learning of a minimal LTL_f formula relative to a given set of examples E^+ and E^- follows the steps in Algorithm 2. For a fixed bound N, we construct an AFA for *some* LTL_f formula of size bounded by N, and that entails all the positive examples, and none of the negative examples. If no such AFA exists, we increment N by one and repeat the process until termination (Theorem 2). In what follows, we show how to reduce the problem of learning an AFA to a *boolean satisfiability* problem.

Algorithm 2 Passive learning of a minimal LTL_f formula.

1: Fix $N = 1$
2: Construct an AFA $\mathcal{A}_N$ for some LTL_f formula φ of size bounded by N that is consistent with the set of examples.
3: If no AFA exists, increment N by one and go to 2.
4: Extract an LTL_f formula φ from the structure of $\mathcal{A}_N$. Return φ.

Theorem 2. *Algorithm 2 returns a minimal LTL_f formula that is consistent with the set of examples.*

Proof. For any given pair of disjoint sets of examples E^+ and E^-, there exists an LTL_f formula that is consistent with them – e.g. $\varphi := (\bigvee_{e \in E^+} toLTL(e)) \wedge (\bigwedge_{e \in E^-} \neg toLTL(e))$, where $toLTL(s_1 \cdots s_n) := \bigwedge_{1 \le i \le n} X^{(i)}(s_i)$. Therefore, Algorithm 2 is guaranteed to terminate. Minimality follows from the guarantees on the size of φ in Step 2. $\square$

5.1 Boolean Satisfiability

The *boolean satisfiability* problem (SAT, for short) relative to a finite set of boolean variables V, and a propositional formula ϕ over variables V is to find an assignment to variables V such that $\phi(V)$ holds true. Such assignment is called a *model*. If no model exists, we say that the problem is *unsatisfiable*. It is useful to represent ϕ in *conjunctive normal form* (CNF), that is, a conjunction of *clauses* where each clause is a disjunction of literals over V. SAT is NP-complete (Cook 1971). Despite its non-tractable complexity, very efficient algorithms exist. In the past, a variety of problems have been solved efficiently via reductions to SAT – e.g. planning, circuit design, and theorem proving (cf. (Biere et al. 2009)).

Example Let $V := \{a, b, c\}$, and $\phi(a, b) := (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a)$. The SAT problem $\langle V, \phi \rangle$ is satisfiable, and has two models: $a = b = c =$ true, and $a = b = c =$ false.

5.2 Reduction into SAT

In this section we address Step 2 in Algorithm 2. Namely, we aim to construct an AFA for some LTL_f formula φ that is consistent with a given set of positive and negative examples. We reduce the task into a SAT problem. Intuitively, (some) variables simulate the structure of an AFA. Some clauses force the AFA to be well constructed, and other clauses force the AFA to be consistent with the examples.

In what follows, we explain the semantics of the variables associated with the structure of the AFA. Then, we provide details on how to enforce well-constructed AFA. Finally, we detail how to enforce acceptance (resp. rejection) of positive (resp. negative) examples. By omitting certain clauses, we relax the constraint on the size of φ. Details follow.

AFA structure We number skeletons s according to a fixed predefined order $1 \le s \le N$ and, by convention, we set the root of the AFA be the skeleton indexed with $s = 1$. The type of operator represented by each skeleton s is indicated by the truth of one of the the variables in the set SkType(s).

$$\text{SkType}(s) := \{\mathsf{AND}(s), \mathsf{OR}(s), \mathsf{NEXT}(s), \mathsf{WNEXT}(s), \mathsf{UNTIL}(s), \mathsf{RELEASE}(s), \mathsf{EVENTUALLY}(s), \mathsf{ALWAYS}(s), \mathsf{LIT}(s)\}$$

We keep track of the AFA structure by means of variables $\mathsf{A}(s, s')$ and $\mathsf{B}(s, s')$. Skeletons s that represent unary operators (NEXT, WNEXT, ALWAYS, EVENTUALLY) have one associated skeleton s' that makes $\mathsf{A}(s, s')$ true. Skeletons s that represent binary operators (AND, OR, RELEASE, UNTIL) have, in addition, one associated skeleton s'' that makes $\mathsf{B}(s, s'')$ true. Variable $\mathsf{LIT}(s)$ is true when skeleton s represents a literal. For example, the truth of $\mathsf{AND}(s)$ indicates that skeleton s represents a conjunctive subformula of the form $\alpha \wedge \beta$. The subformulae α and β are associated with the skeletons s' and s'' that make true $\mathsf{A}(s, s')$ and $\mathsf{B}(s, s'')$. Similar interpretations are given to the other skeleton types.

Enforcing formula size bound An extra set of constraints can be incorporated into the SAT model to enforce that the size of the learned AFA is bounded by N. These constraints have the form $\neg A(s, s') \vee \neg A(s'', s')$ and $\neg B(s, s') \vee \neg B(s'', s')$ and prevent the model from *reusing* skeletons to represent different subformulae with an equivalent syntactic form.

When these constraints are not incorporated into the SAT problem, the size of the AFA (and corresponding LTL_f formula) constructed from the model may be larger than the bound N. This is caused by subformulae that appear duplicated repeated times in the learned model. Allowing the

learned model to reuse syntactically equivalent subformulae produces more compact AFA, and therefore simpler models. Arguably, this may be also a good property to pursue.

Enforcing well-constructed AFA for LTL_f It is known that AFA are strictly more expressive than LTL_f, and not each AFA is a transformation of some LTL_f formula.[1] We constrain the structure of the models to be learned to enforce that the learned AFA is well-constructed, and that corresponds to the transformation of some LTL_f formula.

We require that each skeleton s (recall that $1 \leq s \leq N$, and the AFA is rooted at skeleton $s = 1$) represents either a literal or one, and only one LTL_f operator. In order to avoid cycles, and enforce a well-constructed AFA, we require skeletons s' and s'' have higher index than skeleton s, and s'' have higher index than s'. We accomplish this with the following sets of constraints, where $\mathsf{oneOf}(V)$ is a propositional formula that forces that one, and only one of the variables in V is true.

$$\mathsf{oneOf}(\mathrm{SkType}(s))$$
$$\mathsf{oneOf}(\{\mathsf{A}(s,s') \mid s+1 \leq s' < N\})$$
$$\mathsf{oneOf}(\{\mathsf{B}(s,s'') \mid s+1 < s'' \leq N\})$$
$$\mathsf{oneOf}(\{\mathsf{L}(s,v) \mid 1 \leq v \leq |V|\} \cup \{\mathsf{L}(s,-v) \mid 1 \leq v \leq |V|\})$$

Finally, we do not allow skeletons s with index greater than $N-2$ (resp. $N-1$) to represent binary (resp. unary) operators.

$\neg\,\mathsf{NEXT}(N)$	$\neg\,\mathsf{AND}(N)$	$\neg\,\mathsf{AND}(N-1)$
$\neg\,\mathsf{WNEXT}(N)$	$\neg\,\mathsf{OR}(N)$	$\neg\,\mathsf{OR}(N-1)$
$\neg\,\mathsf{EVENTUALLY}(N)$	$\neg\,\mathsf{UNTIL}(N)$	$\neg\,\mathsf{UNTIL}(N-1)$
$\neg\,\mathsf{ALWAYS}(N)$	$\neg\,\mathsf{RELEASE}(N)$	$\neg\,\mathsf{RELEASE}(N-1)$

Enforcing Acceptance of Positive Examples So far, we have enforced the modular skeleton structure of the AFA being learned to be representative of *some* LTL_f formula. We now show how to enforce that the AFA is consistent with (i.e. accepts) all positive examples.

We use variables $\mathsf{RUN}(e,t,s)$ to monitor a run of the AFA on each positive example e. Intuitively, the truth of $\mathsf{RUN}(e,t,s)$ indicates that there exists a run of the AFA on e at level t labeled with the subformula represented by skeleton s. In particular, we require the variable $\mathsf{RUN}(e,1,1)$ to hold true for each positive example e. The implication clauses listed in Table 1 force that tree expansions of the modular skeleton construction are in correspondence with the runs of the AFA *and* that an accepting run exists. These clauses directly replicate the transition function δ defined in Section 4. Note that we no longer need the variable final, because the size of each example is known.

Enforcing Rejection of Negative Examples The AFA $\mathcal{A}$ to be learned has to reject all negative examples. Recall that $\mathcal{A}$ rejects negative example e iff *none* of the runs of $\mathcal{A}$ on e is accepting. Performing this check is challenging for two reasons. First, for the potentially exponential number of runs that need to be checked for rejection. Second, because checking all runs can only be done *after* the automaton has been learned, not during the learning process. We mitigate for the two challenges above by reasoning on the *dualization* of the AFA $\mathcal{A}_\varphi$ to be learned. The *dual* of an AFA $\mathcal{A}_\varphi$ is a transformation AFA $\mathcal{A}_{\mathsf{dual}(\varphi)}$ that accepts the complement language (cf. Proposition 1).

Proposition 1 (from (Muller and Schupp 1987)). *Let $\mathcal{A}_\varphi = \langle \Sigma, Q, q_0, \delta, F\rangle$ be an AFA transformation of LTL_f formula φ, and let $\mathcal{A}_{\mathsf{dual}(\varphi)}$ be the AFA that results from replacing δ by δ_{dual}, where $\delta_{\mathsf{dual}}(q,s) := \delta(\mathsf{dual}(q),s)$, and*

$\mathsf{dual}(p) := \neg p,\ p \in AP$	$\mathsf{dual}(\alpha \vee \beta) := \mathsf{dual}(\alpha) \wedge \mathsf{dual}(\beta)$
$\mathsf{dual}(\neg p) := p,\ p \in AP$	$\mathsf{dual}(\alpha \wedge \beta) := \mathsf{dual}(\alpha) \vee \mathsf{dual}(\beta)$
$\mathsf{dual}(\mathsf{X}\,\alpha) := \mathsf{N}\,\mathsf{dual}(\alpha)$	$\mathsf{dual}(\alpha\,\mathsf{U}\,\beta) := \mathsf{dual}(\alpha)\,\mathsf{R}\,\mathsf{dual}(\beta)$
$\mathsf{dual}(\mathsf{N}\,\alpha) := \mathsf{X}\,\mathsf{dual}(\alpha)$	$\mathsf{dual}(\alpha\,\mathsf{R}\,\beta) := \mathsf{dual}(\alpha)\,\mathsf{U}\,\mathsf{dual}(\beta)$

Then, $\mathcal{A}_{\mathsf{dual}(\varphi)}$ accepts the complement language of $\mathcal{A}_\varphi$.

We require $\mathcal{A}_{\mathsf{dual}(\varphi)}$ to accept all negative examples. The key point to realize is that we can reason on the runs of $\mathcal{A}_{\mathsf{dual}(\varphi)}$ without actually having to construct it: we only need to construct $\mathcal{A}_\varphi$. Given $\mathcal{A}_\varphi$, a run on $\mathcal{A}_{\mathsf{dual}(\varphi)}$ can be simulated on $\mathcal{A}_\varphi$ by reinterpreting each automaton state (i.e. LTL_f subformula) as its dual. In other words, runs of $\mathcal{A}_{\mathsf{dual}(\varphi)}$ are constructed on the fly, but never stored.

Based on the observation above, we use variables $\mathsf{RUN}(e,t,s)$ to monitor automaton runs on each negative example e. The difference with the approach taken to monitor the runs on *positive* examples is that, now, $\mathsf{RUN}(e,t,s)$ monitors a run *on the dual AFA* $\mathcal{A}_{\mathsf{dual}(\varphi)}$. The implication clauses to enforce $\mathcal{A}_{\mathsf{dual}(\varphi)}$ accepts a negative example e are analogous to the ones in Table 1. As usual, variables in SkType indicate the skeleton type in $\mathcal{A}_\varphi$. The exception is that implication clauses are dualized to simulate runs on $\mathcal{A}_{\mathsf{dual}(\varphi)}$. For example, the implication clauses for a skeleton of type $\alpha \vee \beta$ replicate those of its dual subformula in Table 1, $\alpha \wedge \beta$. Namely,

$$\mathsf{RUN}(e,t,s') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{OR}(s) \wedge \mathsf{A}(s,s')$$
$$\mathsf{RUN}(e,t,s'') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{OR}(s) \wedge \mathsf{B}(s,s'')$$

And the dualization of skeletons that represent variables is:

$$\text{If } v \in e[t]\text{: } \bot \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{LIT}(s) \wedge \mathsf{L}(s,v)$$
$$\text{If } v \notin e[t]\text{: } \bot \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{LIT}(s) \wedge \mathsf{L}(s,\neg v)$$

5.3 Recovering the LTL Formula from the AFA

It remains to see how to extract an LTL_f formula from the learned AFA. Observe that the truth of variables $A(s,s)$ and $B(s,s'')$ describe the tree structure of the AFA, and its associated LTL_f formula can be extracted in linear time in the size of the formula, by recursively replacing node labels (i.e. skeleton types) with subformulae from the root automaton.

6 Hardness Bounds

So far, we have presented algorithms for active and passive learning of LTL_f formulae from input examples. A natural question is: *how many examples are required to learn an LTL_f formula of size N?*

An LTL_f formula of size N can be learned with an exponential number of examples when each new example reduces the number of models that are consistent with respect to the previous set of examples. We say that these examples are *informative*. In particular, the counterexamples output

[1] AFA can accept exactly the regular languages, whereas the expressiveness of LTL_f is reduced to star-free regular expressions (De Giacomo and Vardi 2013).

Subformula	Timestep	Implication clauses for a positive example
$\alpha \wedge \beta$	$1 \le t < \lvert e \rvert$	$\mathsf{RUN}(e,t,s') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{AND}(s) \wedge \mathsf{A}(s,s')$ $\mathsf{RUN}(e,t,s'') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{AND}(s) \wedge \mathsf{B}(s,s'')$
$\alpha \vee \beta$	$1 \le t \le \lvert e \rvert$	$\mathsf{RUN}(e,t,s') \vee \mathsf{RUN}(e,t,s'') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{OR}(s) \wedge \mathsf{A}(s,s') \wedge \mathsf{B}(s,s'')$
$\mathsf{X}\,\alpha$	$1 \le t < \lvert e \rvert$ $t = \lvert e \rvert$	$\mathsf{RUN}(e,t+1,s) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{NEXT}(s) \wedge \mathsf{A}(s,s')$ $\bot \leftarrow \mathsf{RUN}(e,\lvert e \rvert,s) \wedge \mathsf{NEXT}(s) \wedge \mathsf{A}(s,s')$
$\mathsf{N}\,\alpha$	$1 \le t < \lvert e \rvert$ $t = \lvert e \rvert$	$\mathsf{RUN}(e,t+1,s) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{WNEXT}(s) \wedge \mathsf{A}(s,s')$ $\top \leftarrow \mathsf{RUN}(e,\lvert e \rvert,s) \wedge \mathsf{WNEXT}(s) \wedge \mathsf{A}(s,s')$
$\alpha\,\mathsf{U}\,\beta$	$1 \le t < \lvert e \rvert$ $t = \lvert e \rvert$	$\mathsf{RUN}(e,t,s'') \vee (\mathsf{RUN}(e,t+1,s) \wedge \mathsf{RUN}(e,t,s')) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{UNTIL}(s) \wedge \mathsf{A}(s,s') \wedge \mathsf{B}(s,s'')$ $\mathsf{RUN}(e,\lvert e \rvert,s'') \leftarrow \mathsf{RUN}(e,\lvert e \rvert,s) \wedge \mathsf{UNTIL}(s) \wedge \mathsf{B}(s,s'')$
$\alpha\,\mathsf{R}\,\beta$	$1 \le t < \lvert e \rvert$ $1 \le t \le \lvert e \rvert$	$\mathsf{RUN}(e,t,s') \vee \mathsf{RUN}(e,t+1,s) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{RELEASE}(s) \wedge \mathsf{A}(s,s') \wedge \mathsf{B}(s,s'')$ $\mathsf{RUN}(e,t,s'') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{RELEASE}(s) \wedge \mathsf{B}(s,s'')$
$\mathsf{F}\,\alpha$	$1 \le t < \lvert e \rvert$ $t = \lvert e \rvert$	$\mathsf{RUN}(e,t,s') \vee \mathsf{RUN}(e,t+1,s) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{EVENTUALLY}(s) \wedge \mathsf{A}(s,s')$ $\mathsf{RUN}(e,\lvert e \rvert,s') \leftarrow \mathsf{RUN}(e,\lvert e \rvert,s) \wedge \mathsf{EVENTUALLY}(s) \wedge \mathsf{A}(s,s')$
$\mathsf{G}\,\alpha$	$1 \le t < \lvert e \rvert$ $1 \le t \le \lvert e \rvert$	$\mathsf{RUN}(e,t+1,s) \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{ALWAYS}(s) \wedge \mathsf{A}(s,s')$ $\mathsf{RUN}(e,t,s') \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{ALWAYS}(s) \wedge \mathsf{A}(s,s')$
p	$1 \le t \le \lvert e \rvert$ $1 \le t \le \lvert e \rvert$	If $v \notin e[t]$: $\bot \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{LIT}(s) \wedge \mathsf{L}(s,v)$ If $v \in e[t]$: $\bot \leftarrow \mathsf{RUN}(e,t,s) \wedge \mathsf{LIT}(s) \wedge \mathsf{L}(s,\neg v)$

Table 1: Implication clauses induced by a positive example e, where states are interpreted with the closed-world assumption. We write the implication rule $\alpha \leftarrow \beta$ to represent the formula $\alpha \vee \neg\beta$. Timestep parameter t ranges from 1 to $|e|$. Skeleton indexes s range from 1 to $|S|$; s' ranges from $s+1$ to $|S|$; and s'' ranges from $s+2$ to $|S|$. Finally, $v \in V$ represent state variables.

by the oracle in response to equivalence queries are informative. Informative examples can be also constructed from examples for which two potential models disagree.

Theorem 3. *Active learning of an* LTL_f *formula* φ *can be done with a number of queries exponential in the size of* φ.

Theorem 4. *Passive learning of an* LTL_f *formula* φ *can be done with a number of informative examples exponential in the size of* φ.

The bounds in Theorems 3 and 4 contrast with the doubly-exponential bound with respect to the size of the formula required to learn DFA, as noted by Angluin (1987) with the well-known L^* algorithm. The results are encouraging, as learning LTL_f can be realized with exponentially lower bounds on the number of required examples.

7 Practical Applications

In a broad context, model learning of finite-state automata has been used for property extraction and verification of implemented systems. In (Fiterau-Brostean et al. 2017), the authors employed active learning to identify several violations of the standard in SSH implementations. Our methods learn properties in LTL_f. Because LTL_f can be converted into automata (and in fact, we also learn an AFA representation of the formula), our system inherits all benefits of learning automata representations. In particular, LTL_f is equally suitable for verification and model checking. The only caveat is that LTL_f is strictly less expressive than finite-state automata.

Interpretability Learning LTL_f models presents a number of benefits relative to learning automata. Interpretability is one major benefit. Our approach outputs a lifted LTL_f formula that is, in comparison, easier to interpret than grounded models with explicit state representations. Furthermore, LTL_f may be more compact, and easier to interpret than automata – recall that DFA transformations of LTL_f are worst-case doubly exponential, Another benefit is that model maintenance may be easier to perform by a human using LTL_f than using automata. In all cases, it is worth remembering that LTL_f can be transformed into automata if needed, but the opposite is not always true.

Few-shot learning Our approach is *exact*, in the sense that we learn a model that is consistent with *all* given examples. As such, its scalability will, in principle, be more limited than statistical approaches that sacrifice guarantees for the benefit of scalability – e.g. deep learning approaches to time series classification (Karim et al. 2018). On the other hand, we can expect our approach to be able to learn from sets of examples of reasonable size, LTL_f formulae concisely represent properties of state traces, and these properties generalize to unseen examples. For these reasons, our approach can be suitable to few-shot learning.

Behavior Classification We can use LTL_f for multi-class behavior classification. A model can be *trained* to recognize a target behavior, represented by one or more LTL_f formulae, and discriminate it relative to others. The LTL_f formulae act like a *classifier* that is interpretable.

Plan Intent and Recognition Similar to multiclass behavior classification, learned LTL_f models for a multiclass set of behaviors can be used to recognize plan intent. The idea is to use LTL_f models to *monitor* observed behavior and test it against multiple classes. All formulae that are incompatible with the observed behavior are ruled out. The remaining formulae represent possible plan intent.

Target	|AP|	LTL$_f$ learning $\|E^+\|$	$\|E^-\|$	Time	DFA |CS|	S	$\rightarrow$
$\mathsf{X}\, p$	3	1	4	0.3	40	4	32
$p \wedge \mathsf{X}\, q$	3	2	5	0.5	29	4	32
$\mathsf{X}\,(p \wedge q)$	3	5	5	0.4	42	4	32
$\mathsf{N}\,(p \wedge q)$	3	5	6	0.4	29	4	32
$p\, \mathsf{U}\, q$	3	3	4	0.2	22	3	24
$p\, \mathsf{R}\, q$	3	3	3	0.3	22	3	24

Table 2: Number of positive ($|E^+|$) and negative ($|E^-|$) examples needed for active learning of each LTL$_f$ formula, and comparison with the number of characteristic samples (CS) that uniquely define minimal DFA with S states.

Reward Function Learning Another application of learning LTL$_f$ from observation traces is for the purpose of extracting a non-Markovian reward function (Camacho et al. 2017a) from positive and negative observation trace examples. Inferring intended behavior from observation traces is central to inverse reinforcement learning (IRL) (Ng, Harada, and Russell 1999).

Knowledge Extraction What properties does the system under observation have? Our approach can be easily adapted to extract not only one but *all* LTL$_f$ formulae (of bounded size) that are consistent with the observed behavior. It suffices to replace the SAT solver by an all-SAT solver.

LTL mining Another interesting use of our approach is to learn LTL$_f$ formulae that are consistent with examples and have a predefined structure. For example, we may be interested in learning *assume-guarantee* formulae of the form $\mathsf{G}(\varphi_a \rightarrow \varphi_g)$ for some *assumption* subformula φ_a, and some *guarantee* subformula φ_g – e.g. whenever the room temperature surpasses the threshold, the ventilation system is activated. This task has been referred to as LTL *mining*. Lemieux, Park, and Beschastnikh (2015) introduced a method for LTL mining that fulfills finite observation traces with an infinite number of terminal events, from which a consistent LTL formula is learned. Shah et al. (2018) adopted a non-exact, Bayesian approach to infer models in terms of three behaviors encoded as LTL templates. Our approach can be adapted to LTL$_f$ mining by overconstraining the SAT model with the pattern of the formula to be learned. In our example, the first automaton type has to be an *always* (G) operator, and the second automaton type has to be a logical disjunction ($\vee$) – recall that $\mathsf{G}(\varphi_a \rightarrow \varphi_g) \equiv \mathsf{G}(\neg\varphi_a \vee \varphi_g)$.

8 Experiments

To the best of our knowledge, we have presented the first approach to learning LTL$_f$ models from input examples using SAT. We conducted a series of tests to evaluate the benefits of our approaches. For reference, we compared our system with a SAT-based approach to learn DFA (Giantamidis and Tripakis 2016), and with a state-of-the-art deep learning approach for time series classification (Karim et al. 2018).

Our system was implemented in Python, using off-the-shelf SAT solver *Pycosat* (Biere 2008). In our tests, we allowed the learned model reuse subformulae (cf. Section 5.2).

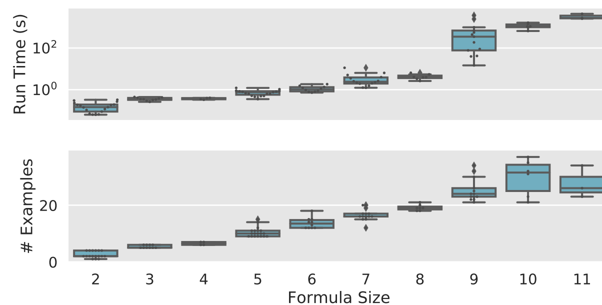


Figure 3: Summary of experimental results on active learning of LTL$_f$. Run time (top) and number of examples needed (bottom) to learn a variety of LTL$_f$ formulae of differing size.

We limited memory to 8GB.

Few-shot learning We implemented an active learner of LTL$_f$ that performs equivalence queries to an oracle. The oracle generates counterexamples as follows. First, we check whether the guessed (φ_G) formula is equivalent to the target (φ_T). This is done by checking emptiness of $\varphi_T \wedge \neg\varphi_G$, and $\neg\varphi_T \wedge \varphi_G$. If the guessed formula is not equivalent to the target, either a positive or negative example is generated (at random, if it exists). Positive examples are generated by searching for an accepting run of the DFA for $\varphi_T \wedge \neg\varphi_G$, which we do via breadth-first search on the automaton. Negative examples are generated similarly with the DFA for $\neg\varphi_T \wedge \varphi_G$. Table 2 summarizes the results of our tests with simple target LTL$_f$ formulae over AP $= \{p, q, r\}$. We compare these numbers with the number of *characteristic samples* (CS) for each target language. A CS is a minimum set of examples (with explicit states) that uniquely defines a minimum DFA (of size S) that models the language (cf. (Giantamidis and Tripakis 2016)). Noteworthy, our approach needed a number of examples that was much lower than the size of the CS. This indicates the potential of our approach in few-shot learning, and the benefits of exploiting lifted state representations.

We conducted a similar test with the SAT-based approach for learning DFA presented in (Giantamidis and Tripakis 2016). Their approach works with *explicit* grounded state representations. Unless the CS was given as input, their system needed a massive number of random examples to learn the target language.

In order to test the scalability of our approach, we constructed a set of formulae with conjunctions of "eventuallys", nesting of "untils", nesting of "releases", industrial patterns from Dwyer, Avrunin, and Corbett (1999), and randomly generated formulae. Figure 3 (right) summarizes the number of examples needed to learn each formula, and the run time of our system. Our active learning system needed a reduced number of examples that increased linearly with the size of the target formula, with relatively low variance. In terms of run time, we observe that formulae with size larger than 8 required significantly more computational effort.

Behavior Classification We conducted experiments to evaluate the adequacy of LTL$_f$ in behavior classification. Given

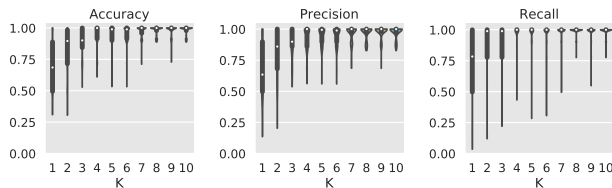


Figure 4: Results on behavior classification across different sizes of the training sets. The training set is constructed by sampling K examples on each of the four behaviors. Results over 100 runs are reported.

a set of examples manifesting different *behavior* in solving different tasks, the objective was to discriminate one target behavior from the others.

We considered the *openstacks* planning benchmark domain (retrieved from (Muise 2016)) as a testbed for generating examples. In the openstaks domain, the agent has to complete and ship a number of orders (in our tests, 5 orders). An order is completed when all its parts are assembled. Different parts from different orders can be stacked and assembled. We modified the domain and designed four behaviors: each behavior ships orders in a particular fixed order. Then, we considered a planning task, common to all behaviors. We generated 1000 examples per behavior. The training set was created by sampling k examples per behavior – those sampled from the target (resp. non-target) behavior constitute the positive (resp. negative) examples. All remaining examples were left to the testing set. Examples were generated using the Top-k planner KStar (Katz et al. 2018), then transformed into state plans, and post-processed to eliminate auxiliary fluents used to enforce desired behavior.

Figure 4 shows the accuracy, precision, and recall of the LTL_f classifiers obtained. We ran 100 experiments per different k. The width of each violin plot is an indicator of the density of datapoints. Our system was able to learn with a few examples per task (K). Most examples had length 25. The formulae learned had size 3 in most cases. Remarkably, those formulae are *not* the formulae that describe the target behavior, but rather simpler formulae that *discriminate* the target behavior from the others given the common constraints of the environment dynamics. For example, the LTL_f formula ((*not* (*shipped o5*))) U(((*stacks_avail n4*))) received full score.

We compared the performance of our system with the state-of-the-art multivariate time series classification system in (Karim et al. 2018). Their system takes a multivariate state representation as input, and uses an LSTM deep neural network architecture with convolutional filters for classification. The learning rate was set to $1e^{-3}$, using Adam optimizer. We trained the model with 20 epochs. Accuracy results obtained for different values of K were similar to those in Figure 4 – e.g. 48% accuracy for $K = 1$; 75% for $K = 2$; 96% for $K = 3$, and 99.8% for $K = 10$. Learning took more than four epochs to reach 100% accuracy in the training set, with 3 to 4 seconds per epoch. In contrast, our system learned LTL_f formulae with considerably less computational effort. For comparison, our system only needed on the order of 0.1 seconds to learn LTL_f formulae.

Interpretability An advantage of our model learning approach is the interpretability of the models with respect to alternative representations. We elaborate on our claim below. DFA tend to be difficult to interpret. Observe that the models in Figure 2 have a very small DFA representation (only three and four states), whereas the number of state transitions (ranging from 24 to 32, indicated with a symbol $\rightarrow$) is too high to be easily interpreted by a human. In contrast, learned LTL_f models resulted in small formulae that are more manageable. Deep learning models, as those obtained with Karim et al. (2018)'s approach, are not human-interpretable. In contrast, our system generates classifiers with associated semantics that are interpretable.

9 Summary and Discussion

We introduced the problems of active and passive learning of an LTL_f formula that captures the temporally extended behavior of a set of positive- and negative-example observation traces. In Section 7, we identified some practical applications of LTL_f learning. In the context of planning, learning LTL_f classifiers has application in plan recognition. Another interesting line of research is in learning temporal invariants, constraints, and heuristics for planning. In non-deterministic and probabilistic planning, landmarks and no-goods can be learned by taking plans obtained from the determinization of the problem and using them as examples – positive if they reach the goal, negative if they reach a deadend.

We presented a SAT-based approach to passive learning of LTL_f formulae, and showed how it can be extended to an active learning system. Our approach proved to be fast and successful at learning models that generalize from few examples, and the learned models were more interpretable than the alternative. Our approach to learning LTL_f has several sources of implicit bias. coming from the preference to learn models (either LTL_f formulae or AFA) of smaller size and from the choice of skeletons, which collectively impose a structural bias by pre-defining the primitive building blocks used to construct LTL_f formulae. More generally, LTL_f mining for a specific pattern can be seen as a form of search bias that imposes hard constraints on the structure of the learned formula. A further source of bias comes from the examples that are generated by the oracle, and the heuristics of the SAT solver. Whereas we leave further investigation to future work, one can imagine that an adequate oracle that outputs examples that optimally disambiguate between the models that are still consistent with the observed examples may help in the learning process.

An obvious restriction of the work presented here is that it does not deal explicitly with noisy data. In future work, we also plan to investigate this setting (c.f. Kasenberg and Scheutz (2017) and Shah et al. (2018) for potential approaches). While noise constrains the applicability of our work, there are numerous dynamical systems that do not manifest noise in their data such as data generated by software systems including user interaction logs, transaction logs, and data from discrete control systems.

Acknowledgements

We thank the reviewers for their thoughtful comments on an earlier version of this paper. We also acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Microsoft Research.

References

Angluin, D.; Eisenstat, S.; and Fisman, D. 2015. Learning regular languages via alternating automata. In *IJCAI*, 3308–3314.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75(2):87–106.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Bacchus, F.; Boutilier, C.; and Grove, A. J. 1996. Rewarding behaviors. In *AAAI*, 1160–1167.

Baier, J. A., and McIlraith, S. A. 2006. Planning with temporally extended goals using heuristic search. In *ICAPS*, 342–345.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2011. Specifying and computing preferred plans. *Artificial Intelligence* 175(7–8):1308–1345.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Biere, A. 2008. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation* 4(2-4):75–97.

Bollig, B.; Habermehl, P.; Kern, C.; and Leucker, M. 2009. Angluin-style learning of NFA. In *IJCAI*, 1004–1009.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017a. Non-markovian rewards expressed in LTL: guiding search via reward shaping. In *SOCS*, 159–160.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017b. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, 3716–3724.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018. Finite LTL synthesis as planning. In *ICAPS*, 29–38.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2018. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, 29–38.

Chandra, A. K.; Kozen, D.; and Stockmeyer, L. J. 1981. Alternation. *Journal of the ACM* 28(1):114–133.

Coles, A., and Coles, A. 2011. LPRPG-P: relaxed plan heuristics for planning with preferences. In *ICAPS*, 26–33.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC*, 151–158.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 854–860.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 1558–1564.

De Giacomo, G.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, 1027–1033.

de la Rosa, T., and McIlraith, S. 2011. Learning domain control knowledge for TLPlan and beyond. In *PAL*, 36–43.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1999. Patterns in property specifications for finite-state verification. In *ICSE*, 411–420.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *ICAPS*, 374–377.

Finkbeiner, B., and Sipma, H. 2004. Checking finite traces using alternating automata. *Formal Methods in System Design* 24(2):101–127.

Fiterau-Brostean, P.; Lenaerts, T.; Poll, E.; de Ruiter, J.; Vaandrager, F. W.; and Verleg, P. 2017. Model learning and model checking of SSH implementations. In *SPIN@ISSTA*, 142–151.

Gabel, M., and Su, Z. 2008. Symbolic mining of temporal specifications. In *ICSE*, 51–60.

Gabel, M., and Su, Z. 2010. Online inference and enforcement of temporal properties. In *ICSE*, 15–24.

Giantamidis, G., and Tripakis, S. 2016. Learning Moore machines from input-output traces. In *FM*, 291–309.

Karim, F.; Majumdar, S.; Darabi, H.; and Harford, S. 2018. Multivariate LSTM-FCNs for time series classification. *CoRR* abs/1801.04503. http://arxiv.org/abs/1801.04503.

Kasenberg, D., and Scheutz, M. 2017. Interpretable apprenticeship learning with temporal logic specifications. In *CDC*, 4914–4921.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *ICAPS*, 132–140.

Lemieux, C.; Park, D.; and Beschastnikh, I. 2015. General LTL specification mining. In *ASE*, 81–92.

Muise, C. 2016. Planning.Domains. In *the 26th International Conference on Automated Planning and Scheduling - Demonstrations*.

Muller, D. E., and Schupp, P. E. 1987. Alternating automata on infinite trees. *Theoretical Computer Science* 54:267–276.

Neider, D., and Gavran, I. 2018. Learning linear temporal properties. In *FMCAD*, 1–10.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations : Theory and application to reward shaping. In *ICML*, volume 3, 278–287.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, 179–190.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*, 46–57.

Shah, A.; Kamath, P.; Shah, J. A.; and Li, S. 2018. Bayesian inference of temporal task specifications from demonstrations. In *NeurIPS*, 3808–3817.

Smetsers, R.; Fiterau-Brostean, P.; and Vaandrager, F. W. 2018. Model learning as a satisfiability modulo theories problem. In *LATA*, 182–194.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *IJCAI*, 1696–1703.

Vaandrager, F. W. 2017. Model learning. *Communications of the ACM* 60(2):86–95.

Vardi, M. Y. 1997. Alternating automata: Unifying truth and validity checking for temporal logics. In *CADE*, 191–206.

Walkinshaw, N.; Derrick, J.; and Guo, Q. 2009. Iterative refinement of reverse-engineered models by model-based testing. In *FM*, 305–320.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic LTLf synthesis. In *IJCAI*, 1362–1369.