Learning Interpretable Classifiers for PDDL Planning

Arnaud Lequen

IRIT, Université Toulouse III - Paul Sabatier
Arnaud.Lequen@irit.fr

Abstract. We consider the problem of synthesizing interpretable models that recognize the behaviour of an agent compared to other agents, on a whole set of similar planning tasks expressed in PDDL. Our approach consists in learning logical formulas, from a small set of examples that show how an agent solved small planning instances. These formulas are expressed in a version of First-Order Temporal Logic (FTL) tailored to our planning formalism. Such formulas are human-readable, serve as (partial) descriptions of an agent's policy, and generalize to unseen instances. We show that learning such formulas is computationally intractable, as it is an NP-hard problem. As such, we propose to learn these behaviour classifiers through a topology-guided compilation to MaxSAT, which allows us to generate a wide range of different formulas. Experiments show that interesting and accurate formulas can be learned in reasonable time.

1 Introduction

One of the main strengths of PDDL planning models is that they are succinct and human-readable, but can nonetheless express general, complex problems, whose state search spaces are exponential in the size of the encoding – as can be the solutions. As a consequence, given a set of examples of the behaviour of an agent (called traces), understanding and recognizing this behaviour can be tedious.

In order to summarize the behaviour of a planning agent in a concise, interpretable way, we propose to learn properties that are specific to the solutions proposed by this agent. Such properties, expressed in a temporal logic tailored to fit PDDL planning models, are not only human-readable, but are also general, and can be evaluated against different instances of the same planning problem. This allows them to recognize the behaviour of an agent on instances that are substantially different from the ones used in the set of examples.

More specifically, the problem we tackle is the one where, given a set of positive example traces (the ones of the agent we seek to recognize) and negative examples traces (the ones of other agents), we wish to learn a model that can discriminate as well as possible between positive and negative traces. A wide variety of techniques and models of different natures have been proposed in the literature. Among these, the learning of finite-state automata (DFA) is a well-studied problem [1, 26, 28], but DFAs can grow quickly (thus becoming harder to interpret) and do not generalize to instances not in the example set. More recently, neural network-based architectures such as LSTMs [29] have shown very promising results, but lack interpretability, and the rationale for their decision is rarely clear.

In the past decade, significant efforts have been made towards learning logical formulas expressed in (a form of) temporal logic. Such works [24, 25, 10, 21, 4, 5] often leverage symbolic methods to learn Linear Temporal Logic (LTL) formulas [23] that fit the example traces, and thus share some similarities with our work. Some other authors propose other techniques, such as Latent Dirichlet Allocation [15], which stems from the field of natural language processing.

However, in all of these cases, the knowledge extracted from the sets of examples has the major drawback of not generalizing well to unknown instances. This is due to the choice of the language used to express these properties. For instance, since LTL formulas are built over a set of propositional variables, they do not generalize to models that do not share the same variables.

To address this issue, we propose to learn properties in a version of First-Order Temporal Logic (FTL). When tailored to the PDDL planning formalism, FTL can express a wide range of properties that generalize from one planning instance to the other, given that they model similar problems. This was shown in [2], who proposed to express *search control knowledge* in a language similar to ours, albeit with the aim of guiding the search of a planner designed to use such knowledge. In [5], the authors proposed to synthesize such control knowledge automatically, and thus address the problem of learning properties expressed in a fragment of FTL.

In this paper, we show that it is possible to learn richer and more expressive properties, using the whole range of FTL operators and modalities. The properties we wish to learn should describe the behaviour of a given planning agent, without being true for the behaviour of other agents. We show that learning such formulas is computationally intractable, as the associated decision problem is NP-hard. This is why the core of our approach consists in encoding the learning problem into a MaxSAT instance, which has the added benefit of showing resilience to any potential noise in the set of training examples. To make the search more efficient, we fix the general topology of the target formula before the encoding. In addition to alleviating the load on the MaxSAT solver and rendering the algorithm more parallelizable, this also increases the diversity in the formulas learned by our algorithm, thus providing varied descriptions of the behaviour of the agent of interest.

Our article is organised as follows: Section 2 introduces the planning formalism as well as the FTL language. Section 3 formally introduces the learning problem we tackle in this paper, and shows that the associated decision problem is intractable. Sections 4 and 5 present some technical choices that we made to solve our problem in reasonable time in practice. In Section 6, we describe our reduction of the problem to MaxSAT, and in Section 7, we present our experimental results, as well as a few examples of formulas that are within reach of our implementation.

2 Background

2.1 Planning with PDDL

This section introduces the model that we use to describe planning tasks. Our definition of a PDDL planning task differs from [11], as we require the organization of the objects of our instances into types. The model we use resembles the one defined in [13]

Definition 1 (Type tree)**.** *A* type tree $\mathcal{T}$ *is a non-empty tree where each node is labeled by a symbol, called a* type*. For any type* $\tau \in \mathcal{T}$*, we call* strict subtype *any descendant* τ' *of* τ*.* τ' *is a subtype of* τ *(denoted* $\tau' \preceq \tau$*) when* τ' *is a strict subtype of* τ *or when* $\tau' = \tau$*.*

Definition 2 (Object class)**.** *Let* $\mathcal{O}$ *be a set of elements called* objects*. We call* object class *any subset of* $\mathcal{O}$*. A class* c_i *is said to be a* subclass *of type* c_j *if* $c_i \subseteq c_j$*.*

Definition 3 (Type hierarchy)**.** *A* type hierarchy $\mathcal{H}$ *over type tree* $\mathcal{T}$ *is a set of object classes such that* $\mathcal{O} \in \mathcal{H}$*, and such that each object class of* $\mathcal{H}$ *is mapped to a unique type of* $\mathcal{T}$*. This mapping* $\tau : \mathcal{H} \to \mathcal{T}$ *is such that for any pair* c_i, c_j *of object classes:*

- c_i *is a subclass of* c_j *iff* $\tau(c_i)$ *is a subtype of* $\tau(c_j)$*;*
- $c_i \cap c_j = \varnothing$ *iff* $\tau(c_i)$ *is not a subtype of* $\tau(c_j)$ *(and conversely).*

We say that object $o \in \mathcal{O}$ *is of type* $\tau(o) := \tau(c)$ *where* c *is the smallest (for inclusion* $\subseteq$*) class of* $\mathcal{H}$ *to which* o *belongs.*

Definition 4 (Predicate, atoms and fluents)**.** *A* predicate p *is a symbol, with which is associated:*

- *An arity* $ar(p) \in \mathbb{N}$
- *A type for each of its arguments. For* $i \in \{1, \ldots, ar(p)\}$*, the type of its argument at position* i *is denoted* $\tau_p(i) \in \mathcal{T}$

An atom *is a predicate for which each argument is associated with a symbol, which can be a variable symbol, or an object of* $\mathcal{O}$*. When the* i*-th argument of the atom is an object* $o \in \mathcal{O}$ *(associated to type hierarchy* $\mathcal{H}$*), then we require that* $\tau(o) = \tau_p(i)$*. The atom consisting of predicate* p *and symbols* $x_1, \ldots, x_{ar(p)}$ *is denoted* $p(x_1, \ldots, x_{ar(p)})$*.*

A fluent *is an atom where each argument corresponds to an object of* $\mathcal{O}$*. A* state *is a set of fluents.*

Definition 5 (Action schema and operators)**.** *An* action schema *is a tuple* $a = \langle \mathit{pre}(a), \mathit{add}(a), \mathit{del}(a) \rangle$*, such that* $\mathit{pre}(a)$*,* $\mathit{add}(a)$ *and* $\mathit{del}(a)$ *are sets of* atoms *instantiated with variables only.*

An operator o *is akin to an action schema, except that the sets* $\mathit{pre}(\mathsf{o})$*,* $\mathit{add}(\mathsf{o})$ *and* $\mathit{del}(\mathsf{o})$ *are sets of fluents.*

Definition 6 (PDDL planning problem)**.** *A PDDL planning problem is a pair* $\Pi = \langle \mathcal{D}, \mathcal{I} \rangle$ *where* $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}, \mathcal{T} \rangle$ *is the domain and* $\mathcal{I} = \langle \mathcal{O}, \mathcal{H}, I, G \rangle$ *is the instance.*

The domain $\mathcal{D}$ *consists of a set* $\mathcal{P}$ *of predicates, a set of actions schemas* $\mathcal{A}$*, and a type hierarchy* $\mathcal{T}$*.*

The instance $\mathcal{I}$ *consists of a set of objects* $\mathcal{O}$ *and an associated type hierarchy* $\mathcal{H}$*, as well as two states,* I *and* G*, which are respectively the initial state and the goal conditions.*

An operator o is applicable in a state s if $\mathsf{pre}(\mathsf{o}) \subseteq s$. The state that results from the application of o in s is $s[\mathsf{o}] = (s \setminus \mathsf{del}(\mathsf{o})) \cup \mathsf{add}(\mathsf{o})$.

A sequence of operators $\mathsf{o}_1, \ldots, \mathsf{o}_n$ is called a *plan* for Π if there exists a sequence of states $s_0, \ldots, s_n$ where $s_0 = I$, and which is such that, for all $i \in \{1, \ldots, n\}$, $s_i = s_{i-1}[\mathsf{o}_i]$ and o_i is applicable in s_{i-1}. Such a sequence of states (which is unique for each plan) is called a *trace*. A plan is called a *solution-plan* if, in addition to this, $G \subseteq s_n$. We say that a fluent $p(o_1, \ldots, o_{ar(p)})$ is true in state s iff $p(o_1, \ldots, o_{ar(p)}) \in s$.

2.2 First-Order Temporal Logic (FTL)

Syntax Let $\mathcal{X}$ be a set of variable symbols, $\mathcal{P}$ a set of predicates, and $\mathcal{T}$ a type tree. We define our language $\mathcal{L}_{\text{FTL}}$ such that:

$$\psi := \exists x \in \tau.\psi \mid \forall x \in \tau.\psi \mid \varphi$$

where $\varphi \in \mathcal{L}_{\text{TL}}$, and $\mathcal{L}_{\text{TL}}$ is such that:

$$\varphi := \top \mid p(x_1, \ldots, x_{ar(p)}) \mid \neg\varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \overline{\bigcirc}\varphi \mid \overline{\Diamond}\varphi \mid \overline{\Box}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi$$

where $x_1, \ldots, x_{ar(p)}$ are variables of $\mathcal{X}$, p a predicate of $\mathcal{P}$, and τ a type. In the following, we will denote $\Lambda = \{\wedge, \vee, \Rightarrow, \mathsf{U}, \bigcirc, \Diamond, \Box\}$ the set of all logical operators. For each operator $\lambda \in \Lambda$, we also note $ar(\lambda) \in \{1, 2\}$ the arity of the operator.

This formulation is akin to Linear Temporal Logic on finite traces (LTL_f) [23], where propositional variables are replaced with first-order predicates and variables. Notice that we only work with formulas in prenex normal form.

Environments The formulas of $\mathcal{L}_{\text{TL}}$ are built on atoms whose arguments are variables of $\mathcal{X}$, while traces contain fluents. We bridge that gap with the notion of *environments*, which are akin to interpretations in first-order logic.

Let us denote $\mathcal{X} = (x_1, \ldots, x_q)$. In addition, let $\mathcal{I}$ be an instance, with objects $\mathcal{O} = \{o_1, \ldots, o_{|\mathcal{O}|}\}$. We call a *partial* environment any assignment of some of the variables $x_1, \ldots, x_q$ to objects of $\mathcal{O}$. Let us denote $\mathsf{var}(e)$ the variables that are assigned an object within the partial environment e. When $\mathsf{var}(e) = \mathcal{X}$, we simply say that e is an environment. We denote any (partial) environment $e = \{x_1 := o_{i_1}, \ldots, x_q := o_{i_q}\}$, where $i_1, \ldots, i_q \in [\![1, |\mathcal{O}|]\!]$.

The object to which variable x is associated to in e is denoted $x[e]$. We also denote $p(x, y)[e]$ the grounding of an atom $p(x, y)$ by an environment e such that $x, y \in \mathsf{var}(e)$. If $e = \{x := o_1, y := o_2, \ldots\}$, then $p(x, y)[e] = p(x[e], y[e]) = p(o_1, o_2)$. By extension, the formula obtained when grounding each atom of φ with e is written $\varphi[e]$.

Semantics Given an environment e, any quantifier-free formula φ of $\mathcal{L}_{\text{TL}}$ can be evaluated against a trace $t = \langle s_0, \ldots, s_n \rangle$, at any step. When $i \in [\![0, n]\!]$, we write $t, e, i \models \varphi$ to denote that formula φ is true at state s_i of trace t with environment e. Temporal modalities, such as $\bigcirc$, $\Diamond$, $\Box$, etc., are used to reason over the states that follow or precede the current state s_i.

$\bigcirc\varphi$ means that property φ is true in the next state, while $\Diamond\varphi$ means that φ is eventually true, in one of the successors of the current state. $\Box\varphi$ means that φ is true from this state on, until the end of the trace, and $\varphi_1 \mathsf{U} \varphi_2$ means that φ_2 is true in some successor state, and until then, φ_1 is true. Operators $\overline{\bigcirc}$, $\overline{\Diamond}$ and $\overline{\Box}$ are the *past* counterparts of the previous connectors: $\overline{\bigcirc}\varphi$ means that φ is true in the previous state, $\overline{\Diamond}\varphi$ that φ is true in some previous state, and $\overline{\Box}\varphi$ that φ is true in every previous state.

To illustrate the language, we introduce the Childsnack problem, which originates from the International Planning Competition (IPC). It consists in making sandwiches and serving them to a group of children, some of whom are allergic to gluten. Sandwiches can only be prepared in the kitchen, and then have to be put on trays, which is the only way they can be brought to the children for service. Among the following FTL formulas, the first indicates that "All children will eventually be served" (and will be satisfied by any solution-plan). The second formula indicates that every sandwich x will eventually be put on some tray, at a moment $t + 1$. For every moment that precedes moment t, x will not be prepared yet (which indicates that the sandwich is actually put on the tray right after being prepared).

$$\forall x \in \text{Child}.\, \Diamond\, \text{served}(x) \quad (1)$$

$$\forall x \in \text{Sandwich}.\, \exists y \in \text{Tray}.\, \text{notprepared}(x)\, \mathsf{U} \bigcirc \text{on}(x, y) \quad (2)$$

Temporal modalities can be expressed in terms of one another. For any quantifier-free formula φ, we have $\Diamond\varphi \equiv \top \mathsf{U} \varphi$, $\Box\varphi \equiv \neg\Diamond\neg\varphi$ and $\overline{\Box}\varphi \equiv \neg\overline{\Diamond}\neg\varphi$. This leads us to an inductive definition of the semantics of our language, for quantifier-free formulas of $\mathcal{L}_{\text{TL}}$:

$$\begin{array}{ll}
t, e, i \models p(x, \ldots, x) & \text{iff } p(x, \ldots, x)[e] \in s_i \\
t, e, i \models \neg\varphi & \text{iff } t, e, i \not\models \varphi \\
t, e, i \models \varphi_1 \wedge \varphi_2 & \text{iff } t, e, i \models \varphi_1 \text{ and } t, e, i \models \varphi_2 \\
t, e, i \models \bigcirc\varphi & \text{iff } i < n \text{ and } t, e, (i+1) \models \varphi \\
t, e, i \models \overline{\bigcirc}\varphi & \text{iff } i > 0 \text{ and } t, e, (i-1) \models \varphi \\
t, e, i \models \overline{\Diamond}\varphi & \text{iff } \exists j \in [\![0, i]\!] \text{ s.t. } t, e, j \models \varphi \\
t, e, i \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff } \exists j \in [\![i, n]\!] \text{ s.t. } t, e, j \models \varphi_2 \\
 & \quad \text{and } \forall k \in [\![i, j-1]\!], t, e, k \models \varphi_1
\end{array}$$

We write $t, e \models \varphi$ as a shorthand for $t, e, 0 \models \varphi$, which means that trace t satisfies the formula φ, since it is true in the initial state of t. A formula $\psi \in \mathcal{L}_{\text{FTL}}$ is evaluated against instantiated traces:

Definition 7 (Instantiated trace). *An* instantiated trace *is a pair $\langle t, \mathcal{I}\rangle$ such that t is a trace where fluents are built on the objects $\mathcal{O}$ of the planning instance $\mathcal{I}$.*

For any partial environment e, $x \in \mathcal{X}$, $o \in \mathcal{O}$, we denote $e[x := o]$ the environment identical to e, but where variable x is associated o. The semantics of $\mathcal{L}_{\text{FTL}}$ is defined as follows:

$$\begin{array}{ll}
\langle t, \mathcal{I}\rangle, e \models \forall x \in \tau.\psi & \text{iff for all } o \in \mathcal{O} \text{ s.t. } \tau(o) = \tau, \\
 & \quad \langle t, \mathcal{I}\rangle, e[x := o] \models \psi \\
\langle t, \mathcal{I}\rangle, e \models \exists x \in \tau.\psi & \text{iff there exists } o \in \mathcal{O} \text{ s.t. } \tau(o) = \tau, \\
 & \quad \langle t, \mathcal{I}\rangle, e[x := o] \models \psi \\
\langle t, \mathcal{I}\rangle, e \models \varphi & \text{iff } t, e \models \varphi
\end{array}$$

where x is a variable, and φ is a formula of $\mathcal{L}_{\text{TL}}$ (thus quantifier-free). We will often denote $\langle t, \mathcal{I}\rangle \models \psi$ as a shorthand for $\langle t, \mathcal{I}\rangle, \varnothing \models \psi$, where $\varnothing$ is the empty environment.

Note that it is well known that the past modalities do not change the expressivity of LTL. As a consequence, our language could have expressed the same properties without modalities $\overline{\bigcirc}$, $\overline{\Diamond}$ or $\overline{\Box}$. However, we include these modalities in our language as they may make some properties more succinct to express [19].

2.3 The MaxSAT problem

Let Var be a set of propositional variables. The boolean satisfiability problem (SAT) is concerned with finding a valuation that satisfies a propositional formula ϕ. Propositional formulas are defined as follows, where $x \in$ Var is a propositional variable:

$$\phi := \top \mid x \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi$$

The maximum boolean satisfiability problem (MaxSAT) is a variant of SAT, in which a valuation of the variables Var of a set of formulas $\{\phi_1, \ldots, \phi_n\}$ is sought. Each formula ϕ_i is assigned a *weight* $w(\phi_i) \in \mathbb{R} \cup \{\infty\}$. The MaxSAT problem consists in finding a valuation v of Var such that the sum of the weights of the formulas that are not satisfied by v is minimal.

3 The $\mathcal{L}_{\text{FTL}}$ learning problem

3.1 Problem definition

Score function Our problem takes in input a *score function*, denoted $\sigma : \mathcal{T} \to \mathbb{R}$, where $\mathcal{T}$ is the set of traces. This function allows us to express preferences on which traces are the most important to capture in the output formula, and which traces are the most important to avoid. In the rest of this article, we will say that an instantiated trace $\langle t, \mathcal{I}\rangle$ is *positive* iff $\sigma(\langle t, \mathcal{I}\rangle) \geq 0$. Otherwise, the instantiated trace is said to be *negative*.

In the following, we use $[\langle t, \mathcal{I}\rangle \models \psi]$ as a shorthand for the function equal to 1 if $\langle t, \mathcal{I}\rangle \models \psi$ and equal to 0 otherwise. The score function generalizes to formulas $\psi \in \mathcal{L}_{\text{FTL}}$ as follows:

$$\sigma^{\mathcal{T}}(\psi) = \sum_{\langle t, \mathcal{I}\rangle \in \mathcal{T}} \sigma(\langle t, \mathcal{I}\rangle)\, [\langle t, \mathcal{I}\rangle \models \psi]$$

Problem 1. *$\mathcal{L}_{FTL}$ learning*

Input: *$\mathcal{D}$ a domain*
$\mathcal{T}$ a set of instantiated traces
$r \in \mathbb{N}$ the maximum number of logical operators in the output formula
$q \in \mathbb{N}$ the maximum number of quantifiers
$\sigma : \mathcal{T} \to \mathbb{R}$ a function called the score function

Output: *A formula $\psi \in \mathcal{L}_{FTL}$ such that ψ has at most r logical operators, and q quantifiers, and*
$\sum_{\langle t, \mathcal{I}\rangle \in \mathcal{T}} \sigma(\langle t, \mathcal{I}\rangle)\, [\langle t, \mathcal{I}\rangle \models \psi]$
is maximal

3.2 Complexity

The decision problem associated to the $\mathcal{L}_{\text{FTL}}$ learning problem is the problem for which the output is *Yes* iff *there exists a formula $\psi \in \mathcal{L}_{FTL}$ satisfying the requirements above, and with score $\sigma^{\mathcal{T}}(\psi) \geq \ell$*, where ℓ is given in input. The proof of intractability consists in a reduction from the NP-hard Set Cover Problem [14]. It is sketched below, and a more detailed proof can be read in the supplementary material of this article [18].

Proposition 1. *The decision problem associated to the $\mathcal{L}_{FTL}$ learning problem is NP-hard*

Problem 2. *Set Cover*

Input: *A set $U = \{1, \ldots, n\}$*
A set S of subsets: $S = \{S_1, \ldots, S_m\} \subseteq 2^U$
$k \in \mathbb{N}$

Output: *Yes* iff *there exists a subset $T \subseteq S$ such that $|T| \leq k$ and $\cup_{s \in T} s = U$*
No otherwise

Proof of Proposition 1 (Sketch). Let us consider an instance of *Set Cover*. We build an instance of *$\mathcal{L}_{FTL}$ learning* that is positive (i.e. outputs *Yes*) *iff* the *Set Cover* instance is positive.

The proof consists in showing that a set of positive traces can be described by a formula $\mathcal{L}_{\text{FTL}}$ satisfying the constraints in input iff there exists a set cover of size at most k. Each of the positive traces is associated to an element j of U, and contains a single state (and thus, temporal modalities have no effect). This single state carries the information as to which subsets contain j. The information consists of fluents of the form $\text{in}(S_i)$. For instance, if only sets $S_1, S_3 \in S$ contain element j, then the j-th trace is $\langle\{\text{in}(S_1), \text{in}(S_3)\}\rangle$.

Each subset S_i of S, in the Set Cover instance given in input, is associated a unique type Set_i in the PDDL domain we build. As our $\mathcal{L}_{\text{FTL}}$ formulas quantify on these types, and since we restrict the number of quantifiers to $q = k$, any output formula of the $\mathcal{L}_{\text{FTL}}$ learning problem can only reason on at most k subsets among those in S. If k quantifiers are enough to distinguish between the positive traces (associated to elements $j \in U$) and a mock trace, then k subsets are enough to cover all elements of U. Otherwise, no set cover of size at most k exists.

More formally, let us define the input of our *$\mathcal{L}_{FTL}$ learning* instance as follows: the domain is $\mathcal{D} = \langle\mathcal{P}, \mathcal{A}, \mathcal{T}\rangle$, where types are $\mathcal{T} = \{\text{Set}_1, \ldots, \text{Set}_m, \text{Set}\}$, $\mathcal{P} = \{\text{in}\}$ (with $ar(\text{in}) = 1$ and $\tau_{\text{in}}(1) = \text{Set}$), and $\mathcal{A} = \varnothing$ (as all traces have length 1, no action is needed).

Each instantiated trace is associated its own instance, which is in turn associated to a unique element $j \in U$: for $j \leq n$, $\mathcal{I}_j = \langle\mathcal{O}, \mathcal{H}, I_j, G_j\rangle$. The shared set of objects contains the sets of the Set Cover input instance, and is such that $\mathcal{O} = \{S_1, \ldots, S_m, d\}$ (where d is a mock object that we use to discriminate traces later). The type hierarchy $\mathcal{H} = \{\{S_i\} \mid S_i \in S\} \cup \mathcal{O}$ associates each set to its own unique type in our PDDL domain: types are such that $\tau(S_i) = \text{Set}_i$ for all $i \leq m$, and $\tau(\mathcal{O}) = \text{Set}$. The information regarding which sets j belongs to is encoded in the initial state and goal of $\mathcal{I}_j$: for $j \leq n$, $I_j = G_j = \{\text{in}(S_i) \mid j \in S_i\}$.

We also define a mock, negative instance $\mathcal{I}_d = \langle\mathcal{O}, \mathcal{H}, I_d, G_d\rangle$, with $I_d = \{\text{in}(d)\}$, and $G_d = I_d$. We have $\mathcal{T} = \{\langle t_j, \mathcal{I}_j\rangle \mid j \leq n\} \cup \{\langle t_d, \mathcal{I}_d\rangle\}$, where $t_j = \langle I_j\rangle$ and $t_d = \langle I_d\rangle$. We set $r = m - 1$, $q = k$, and the score $\sigma(\langle t_j, \mathcal{I}_j\rangle)$ of every trace to 1, except $\langle t_d, \mathcal{I}_d\rangle$ for which $\sigma(\langle t_d, \mathcal{I}_d\rangle) = -1$. The decision problem consists in finding whether there exists a formula $\psi \in \mathcal{L}_{\text{FTL}}$ that is satisfied by every trace, except $\langle t_d, \mathcal{I}_d\rangle$. Thus, the score threshold we set is $\ell = m$. Let us now show that this instance is positive iff our *Set Cover* instance is positive.

Suppose that there exists a set cover $T = \{S_{i_1}, \ldots, S_{i_k}\}$. Then there exists a formula $\psi \in \mathcal{L}_{\text{FTL}}$ with score exactly m:

$$\psi := \exists x_1 \in \text{Set}_{i_1}, \ldots, \exists x_k \in \text{Set}_{i_k}. \bigvee_{j \leq k} \text{in}(x_j)$$

We have that $\langle t_d, \mathcal{I}_d\rangle \not\models \psi$, but all other traces satisfy ψ, since T is a set cover. So the formula has score exactly m, and is a positive instance of *$\mathcal{L}_{FTL}$ learning*.

Now suppose that there exists no set cover of size at most k for the input instance. By contradiction, suppose that there exists a formula ψ with score m. We denote T' the set of types on which ψ quantifies upon. We have that $|T'| \leq k$. Since there exists no set cover of size at most k, then there exists $w \leq n$ such that $w \notin \cup_{s \in T'} s$. The rest of the proof consists in showing that if $\langle t_w, \mathcal{I}_w\rangle \models \psi$, then $\langle t_d, \mathcal{I}_d\rangle \models \psi$, which contradicts the fact that ψ has score m. This can be proven by structural induction on ψ.

Since the $\mathcal{L}_{\text{FTL}}$ learning instance we built is positive iff the set cover instance is positive, the $\mathcal{L}_{\text{FTL}}$ learning problem is NP-hard. $\square$

In [7, 20], the authors tackle the $\mathcal{L}_{\text{LTL}}$ learning problem, where $\mathcal{L}_{\text{LTL}}$ is the fragment of $\mathcal{L}_{\text{TL}}$ without past modalities. Along with other past modalities-free fragments, they showed the corresponding decision problem to be NP-complete.

Given an environment e, a trace t, and a formula $\varphi \in \mathcal{L}_{\text{TL}}$, checking that $t, e \models \varphi$ can be done in space polynomial in $|t|$, $|e|$ and $|\varphi|$ (ex. [8]). The model-checking of $\psi \in \mathcal{L}_{\text{FTL}}$ against some $\langle t, \mathcal{I}\rangle$ can be done by enumerating all relevant environments $e \in \mathcal{O}^q$, and checking that $t, e \models \varphi$, where φ is the quantifier-free part of ψ. As a consequence, the $\mathcal{L}_{\text{FTL}}$ learning problem is in PSPACE. Even though this shows membership, the potential PSPACE-hardness of our problem is still an open problem.

4 Planning problem preprocessing

We present in this section the transformations we bring to the PDDL planning problem before it is passed to our algorithm for learning $\mathcal{L}_{\text{FTL}}$ formulas.

Predicate splitting Each predicate is split into several predicates of size 2, in order to curb the number of fluents while conserving the links between pairs of objects. This allows us to synthesize formulas containing predicates of high arity, while keeping the number of quantifiers of the formula low.

Concretely, a predicate of the form $p(x, y, z)$ will be split into newly-created predicates $p_{12}(x, y)$, $p_{13}(x, z)$, and $p_{23}(y, z)$. Predicate splitting leads to significantly fewer fluents than if the task was to be grounded as is: for a predicate of arity $n \geq 2$, to be grounded with instance $\mathcal{I}$, there are $O(n^2|\mathcal{O}|^2)$ associated fluents, while there would be $O(|\mathcal{O}|^n)$ if the predicate was not split.

Even though the planning model thus obtained is less rich than the original one, we argue that predicate splitting allows us to learn formulas that would be otherwise out of computational reach.

Goal predicates In order to allow the learnt formulas to reason on the goal state, we introduce *goal predicates*. For every predicate $p \in \mathcal{P}$, we introduce the predicate p^G. Then, for each instance $\mathcal{I}$, we introduce the *latent state* $s_{\mathcal{I}}$, which is intuitively a set of fluents that are true in every state of every trace associated to $\mathcal{I}$.

For every fluent $p(o_1, \ldots, o_{ar(p)})$ of the goal state G of $\mathcal{I}$, we add the fluent $p^G(o_1, \ldots, o_{ar(p)})$ to $s_{\mathcal{I}}$.

5 Topology-based guiding

TL chains An interesting representation for formulas φ of $\mathcal{L}_{\text{TL}}$ is a representation as *TL chains*. They are the adaptation to our language of the notion of chain [16, 25], which is useful for representing formulas of modal or propositional logic.

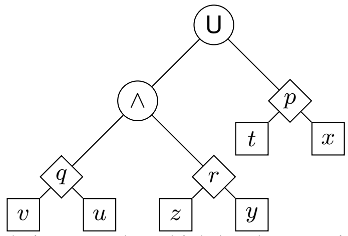


Figure 1: A TL chain example, which has been assigned symbols to its nodes. It represents the formula $(q(v, u) \wedge r(z, y)) \mathsf{U}\, p(t, x)$

A *TL chain* is a Directed Acyclic Graph (DAG) which has three types of nodes: logical connector nodes (represented as $\circ$ in the example of Figure 1), predicate nodes (represented as $\diamond$) and variable nodes (represented as $\square$). In order to represent a correct $\mathcal{L}_{\text{TL}}$ formula, logical connector nodes can only be children of logical connector nodes, predicate nodes children of logical connector nodes, and variable nodes children of predicate nodes. We also impose that every leaf is a variable node. In addition, to stay consistent with the choices we made in Section 4, we only work with TL chains that are binary trees, whose inner nodes have exactly two children.

By assigning a symbol of the correct type (i.e., a logical connector, a predicate symbol or a variable) to each node, we end up with a representation of a $\mathcal{L}_{\text{TL}}$ formula, as illustrated in Figure 1.

For each connector node i of the TL chain, we will denote $succ_L(i)$ (resp. $succ_R(i)$) the left (resp. right) child of node i. It is guaranteed to exist, but might sometimes be a predicate node. In the case of connectors $\alpha \in \{\neg, \bigcirc, \overline{\bigcirc}, \Diamond, \overline{\Diamond}, \Box, \overline{\Box}\}$ that have arity 1, we will use the convention that the value of the right successor is ignored (and will not appear in the $\mathcal{L}_{\text{FTL}}$ formula that ensues), and the left successor will be the root of the formula under the operator α.

In order to alleviate the pressure on the MaxSAT solver, we impose the topology of the output quantifier-free formula before encoding the problem into a propositional formula. This idea was first introduced in [25], in the case of a search for LTL formulas. We also fix the quantifiers of the formula before the encoding, as well as the associated types. All that is left to the MaxSAT solver is to "fill in the blanks" in the TL chains that it is given, so that the associated $\mathcal{L}_{\text{FTL}}$ formula fits the input as well as possible.

Quantifiers In the rest of the article, for practical reasons, we restrict ourselves to learning formulas of the form $\forall x_1 \cdots \forall x_b \exists x_{b+1} \cdots \exists x_q \varphi$, where φ is a formula of $\mathcal{L}_{\text{TL}}$, for which every argument of every predicate is a variable x_i. This allows us to curb the size of the MaxSAT encoding.

6 Reduction to MaxSAT

6.1 Learning algorithm

Algorithm 1 summarizes the procedure that we use to learn $\mathcal{L}_{\text{FTL}}$ formulas out of our input. The subroutines work as follows: gen_TLchains(r) enumerates every TL chain having exactly r connectors. gen_quantifiers(q) enumerates sequences of quantifier symbols of size q, such that all universal quantifiers $\forall$ appear before existential quantifiers $\exists$. gen_types($\mathcal{D}$, q) enumerates every q-combination of types in the type tree $\mathcal{T}$ of $\mathcal{D}$. Finally, the main subroutine, find_formula($\mathcal{D}$, T, ρ, $\{Q_i\}$, $\{\tau_i\}$, σ), encodes the problem of finding an $\mathcal{L}_{\text{FTL}}$ formula fitting the instantiated traces of T, with the constraints imposed by the TL chain ρ, the quantifiers $\{Q_i\}$, and the types $\{\tau_i\}$. find_formula then returns (one of) the best formula(s) it finds, or the token FAIL if none is found.

Algorithm 1: $\mathcal{L}_{\text{FTL}}$ learning

```
Input: Domain D, traces T, parameters r, q, and function σ
Output: A set of L_FTL formulas
found_formulas := [ ]
for ρ ∈ gen_TLchains(r) do
    for Q_1, ..., Q_q ∈ gen_quantifiers(q) do
        for τ_1, ..., τ_q ∈ gen_types(D, q) do
            ψ ← find_formula(D, T, ρ, {Q_i}, {τ_i}, σ);
            if ψ ≠ FAIL then found_formulas.add(ψ);
return found_formulas
```

6.2 Preliminaries to the encoding

Variables Our MaxSAT encoding is built on the set of variables that follows. When possible, we use the following conventions, as closely as possible: nodes of the FL-chain are denoted by i when they are logical connectors (represented by $\bigcirc$ in Figure 1), by ℓ when they are predicate nodes (represented by $\Diamond$), and by v when they are first-order variable nodes (represented by $\Box$). A trace is denoted by t, and a position in this trace is denoted by k (i.e., the k-th state). Moreover, j is an index for a variable of the quantifiers, and p is a predicate.

This leads us to the following variables, as will be used in the MaxSAT encoding. Greek letters denote decision variables while latin characters are for "technical" variables.

- $y_i^{t,k}[e]$: In position k of trace t, with environment e, the formula rooted at node i is true.
- $\delta_{j,v}^{\ell}$: The v-th variable of predicate node ℓ is the variable of quantifier j.
- θ_ℓ^p: The predicate of node ℓ is p.
- λ_i^q: The logical connector at node i is q.
- s_t: Trace t is currently satisfied by the first order formula

"Exactly one" constraints In the encoding of a problem into SAT, some situations require that *exactly one* variable, out of a set of variables, is true. Efficient encodings have been widely studied: see for instance [12, 22]. In the following, we will denote $\mathsf{ExactlyOne}_{s \in S}(v_s)$ the set of propositional constraints enforcing that exactly one of the variables of $\{v_s \mid s \in S\}$ is true.

6.3 Core constraints

Some of the constraints below are adapted from [10, 25, 21], which are concerned with LTL. Our main contribution is the adaptation of the encoding to our language $\mathcal{L}_{\text{FTL}}$, which differs from LTL by its tighter links with PDDL planning models through first-order components.

In the following, we suppose that an empty TL chain ρ has been computed, and that the associated quantifiers and types have been decided. We will denote n its number of connector nodes, and m its number of predicate nodes. As a consequence, there are $2m$ variable nodes. As previously, the number of quantifiers is denoted q. The first $b \leq q$ quantifiers are universal, while the others are existential.

We also suppose that the types on which the quantifiers range, denoted $\tau_1, \ldots, \tau_q$, are already chosen. As a consequence, in this section, the set of relevant environments for instance $\mathcal{I}_t$ associated to trace t, denoted $E_{\mathcal{I}_t}$, only consists of environments of the form $\{x_u := o_u\}_{1 \leq u \leq q}$ where, $\tau(o_u) = \tau_u$, for $u \in [\![1, q]\!]$.

Syntactic constraints This section describes the constraints that ensure that the formula is syntactically well-formed.

The following constraints respectively ensure that every logical connector node has exactly one logical connector assigned, that every predicate node has exactly one predicate, and that each argument of each predicate is bound to a variable on which the formula quantifies.

$$\bigwedge_{i \leq n} \mathsf{ExactlyOne}_{c \in \Lambda}(\lambda_i^c) \wedge \bigwedge_{\ell \leq m} \mathsf{ExactlyOne}_{p \in \mathcal{P}}(\theta_\ell^p)$$

$$\bigwedge_{\ell \leq m} \bigwedge_{s \in \{1,2\}} \mathsf{ExactlyOne}_{j \leq b}(\delta_{j,s}^{\ell})$$

Semantic constraints These constraints ensure that the formula found by the solver is consistent with the traces, and is reminiscent of the model-checking algorithm for modal logic.

The following clauses ensure that the formula ψ that is synthesized is consistent with the traces of T. This is made in accordance with the environments imposed by the quantifier, which are iterated upon. The variable s_t is true iff for every required environment e, $\varphi[e]$ is satisfied by t (where $\varphi[e]$ is the evaluation of formula φ in environment e, and φ is the quantifer-free part of the formula we synthesize). Thus, for every trace $t \in T$, we add the following:

$$s_t \Leftrightarrow \bigwedge_{\substack{o_1 \in \mathcal{O}_1 \\ \cdots \\ o_k \in \mathcal{O}_k}} \bigvee_{\substack{o_{k+1} \in \mathcal{O}_{k+1} \\ \cdots \\ o_q \in \mathcal{O}_q}} y_1^{t,1}[\{x_u := o_u\}_{1 \leq u \leq q}] \tag{3}$$

The following constraints ensure that formulas that consist of a single literal (i.e., a positive or negative fluent) are consistent with the y variables, that give the truth value of a trace at a certain position in the trace, at each node of the TL chain.

Such constraints appear once for every trace $t \in T$, for every position $k \leq |t|$ of this trace, for every predicate node $\ell \leq m$ and every predicate $p \in \mathcal{P}$, for every pair of quantifiers (positions) $j_1, j_2 \leq q$, and for each relevant environment $e \in E_{\mathcal{I}_t}$.

$$\theta_\ell^p \wedge \delta_{j_1,1}^\ell \wedge \delta_{j_2,2}^\ell \Rightarrow \begin{cases} y_\ell^{t,k}[e] & \text{if } t[k] \models p(x_{j_1}, x_{j_2})[e] \\ \neg y_\ell^{t,k}[e] & \text{otherwise} \end{cases} \tag{4}$$

Constraints (5) to (8) appear once for each connector node $i \leq n$ of the formula, each position $k \leq |t|$ of each trace $t \in T$, and for each environment $e \in E_{\mathcal{I}_t}$. They ensure that the logical operators are correctly interpreted.

In the case where the logical connector at node i is a negation $\neg$, or $\Delta \in \{\wedge, \vee, \Rightarrow\}$, or the next operator $\bigcirc$, we have:

$$\lambda_i^\neg \Rightarrow \left(y_i^{t,k}[e] \Leftrightarrow \neg y_{succ_L(i)}^{t,k}[e]\right) \tag{5}$$

$$\lambda_i^\Delta \Rightarrow \left(y_i^{t,k}[e] \Leftrightarrow \left(y_{succ_L(i)}^{t,k}[e] \;\Delta\; y_{succ_R(i)}^{t,k}[e]\right)\right) \tag{6}$$

$$\lambda_i^\bigcirc \Rightarrow \left(y_i^{t,k}[e] \Leftrightarrow y_{succ_L(i)}^{t,k+1}[e]\right) \tag{7}$$

with the convention that $y_{succ_L(i)}^{t,|t|+1}[e]$ is replaced by $\perp$ during the encoding itself. In the case of the finally operator $\Diamond$:

$$\lambda_i^\Diamond \Rightarrow \left(y_i^{t,k}[e] \Leftrightarrow \bigvee_{\substack{k' \\ k \leq k' \leq |t|}} y_{succ_L(i)}^{t,k'}[e]\right) \tag{8}$$

The case of the temporal operators $\Box$, $\overline{\bigcirc}$, $\overline{\Diamond}$, $\overline{\Box}$ and U can be encoded in a way that is similar to the constraints above.

Well-formed fluents constraints The following constraints ensure that, in the output formula ψ, there is a consistency between the types of the variables and the arguments of predicates are assigned to. In other words, when a variable x of type τ is chosen to be the v-th argument of a predicate p that occurs in ψ, we require that $\tau = \tau_p(v)$. This can be done through the following constraints:

$$\bigwedge_{j \leq q} \bigwedge_{\ell \leq m} \bigwedge_{p \in \mathcal{P}} \bigwedge_{j \leq q} \bigwedge_{\substack{v \leq 2 \\ \tau_p(v) \neq \tau_j}} \neg\theta_\ell^p \vee \neg\delta_{j,v}^\ell \tag{9}$$

Weights for the MaxSAT solver Recall that we wish to find a formula ψ that maximizes the function given in Problem 1. The objective of the MaxSAT solver is to minimize the total weight of the falsified soft clauses. As such, for each instantiated trace $\langle t, \mathcal{I}\rangle$, we add the clause s_t, with weight $\sigma(\langle t, \mathcal{I}\rangle)$. This penalizes formulas that falsify traces with a positive score, while rewarding formulas that falsify traces with a negative score.

Pruning non-discriminatory formulas With a given configuration of TL chain, quantifiers and types, it is not guaranteed that there exists a formula ψ that captures (some of) the positive traces while falsifying (some of) the negative traces. To prevent tautology or unsatisfiable formulas from occurring, we enforce the constraint that at least one positive trace is captured and one negative trace is not.

6.4 *Formula quality enhancement*

The constraints of this section filter the solutions so that less interesting formulas, or formulas that could be computed by a run of our algorithm with smaller parameters, are barred from being output.

Syntactic redundancies prevention These constraints prevent idempotent and involutive modalities and operators from being chained in the output formula. These include the negation $\neg$, as well as the temporal operators $\Diamond$ (for which $\Diamond\Diamond\varphi \equiv \Diamond\varphi$) and $\Box$ (which is, likewise, idempotent).

In addition, we prevent redundancies of the form $p(x, y) \,\Delta\, p(x, y)$, where $\Delta \in \{\wedge, \vee, \mathsf{U}, \Rightarrow\}$ is a binary operator. In every case, there exists a smaller (sub-)formula that can be found and that expresses the same property, without the redundant atom. For space reasons, we skip the presentation of the constraints.

Variable visibility We wish to ensure that every variable that we quantify upon in the output formula ψ also appears in an atom of ψ. Otherwise, an equivalent formula could be found by running the algorithm with fewer quantifiers. This is why we force each variable to appear at least once in some atom.

7 Experiments

We implemented Algorithm 1 in Python 3.10, using the MaxSAT solver Z3 [6]. Experiments were conducted on a machine running Rocky Linux 8.5, powered by an Intel Xeon E5-2667 v3 processor, with a 9-hours cutoff and using at most 8GB of memory per run. The code of our implementation and our data are available online [17]. Additional, more detailed test results can also be found in [17].

Even though we often managed to quickly find a formula that perfectly captures the set of examples, we let the algorithm run to the end, so that all TL chains and combinations of quantifiers and types are enumerated.

Building the data sets To assess the performances of our algorithm, we considered domains from the International Planning Competition (IPC), 2 of which are described in Section 7.1. For each of these domains, we generated 23 instances that model problems with similar goals. Then, for each domain, we designed three domain-specific planners, that solve the tasks in a distinctive way.

We built our training sets by selecting 3 small planning instances of each domain, and the associated traces for each planner – for a total of 9 traces per domain. We then created the tasks of finding a formula recognizing the behaviour of each planner, out of 1, 2 or 3 of the training instances. The 20 remaining instances (and their associated plans) were used in the test set. The traces of our training set have length 5 to 21, with an average of 11.8 states.

7.1 *Examples of learnt formulas*

Childsnack We designed three different agents that solve Childsnack instances, as introduced in Section 2.2. Agents NGF and NGL compute solution plans of minimal size, and differ in that agent NGF makes sandwiches with *no gluten first*, and agent NGL makes sandwiches with *no gluten last*. Both agents make all sandwiches, put them on a tray, then serve the children. Agent GS greedily serves children: as soon as a sandwich is made, it is put on a tray and brought to a child. It also prioritizes gluten-free sandwiches.

For each of these behaviours, the total computation time and the total number of formulas found are depicted in Table 3 and Table 2, respectively. As shown on Table 1, we manage to learn formulas that

Table 1: Proportion of traces correctly classified by the most accurate formula on our test set, expressed in percentage (%). Each table presents a different agent solving a set of Childsnack problems. *# ins.* indicates the number of instances in the training set, q the number of quantifiers allowed, and $|\varphi|$ the number of logical operators allowed. Dots correspond to configuration where the run did not terminate.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	67	100	67	100	67	100
3	100	100	100	100	100	100
4	.	100	.	100	.	.

NGF

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	67	67	67	67	67	67
3	100	100	100	100	100	67
4	.	.	.	.	.	.

NGL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	100	100	100	100	100	100
3	100	100	100	100	100	100
4	.	.	.	.	.	.

Table 2: Total number of formulas found for Childsnack.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	22	110	22	112	22	123
3	92	624	92	624	92	671
4	.	746	.	757	.	.

NGF

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	23	126	23	126	23	140
3	92	670	92	679	94	723
4	.	797	.	858	.	.

NGL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	7	76	7	78	8	92
3	79	604	79	681	82	725
4	.	784	.	845	.	.

perfectly capture the test set, even when the number of examples is very limited, and when few quantifiers and logical operators are allowed. Formula 2, given in Section 2.2, is an example of a concise formula that describes the policy of agent GS perfectly. Other formulas include the following:

$$\forall x \in \text{Kitchen}.\ \exists y \in \text{Tray}.\ \Diamond(\text{at}(y, x) \wedge \overline{\Diamond}\neg\text{at}(y, x)) \tag{10}$$

Formula (10) expresses that agent GS eventually comes back to the (only) kitchen with some tray y, even though the tray was brought out of the kitchen at some point in the past. The formula perfectly captures our test set, but does not perfectly capture our *training* set. Indeed, the smallest instance of our training set contains as many children as there are trays, and thus, no tray has to be brought back to the kitchen. Our use of a reduction to MaxSAT allows us to be resilient to this kind of edge cases, and the formula above is satisfactory despite not perfectly fitting the training set.

Spanner Instances of the Spanner domain involve an operator that has to go from a shed to a gate to tighten some nuts, passing through a sequence of locations where single-use spanners can be picked up. Once a location is left, it can not be returned to. Thus, collecting enough spanners before reaching the gate is essential.

Table 3: Total computation times for Childsnack (hh:mm:ss). Rows correspond to the number of allowed logical connectors. Dots correspond to configuration where the run did not terminate.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:08	0:05:03	0:00:08	0:16:43	0:01:14	0:44:48
3	0:00:35	0:19:53	0:00:35	1:07:27	0:06:46	3:35:18
4	.	0:25:27	.	1:32:07	.	.

NGF

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:08	0:05:08	0:00:08	0:17:07	0:01:09	0:47:51
3	0:00:37	0:20:13	0:00:37	1:11:13	0:07:05	3:44:11
4	.	0:26:29	.	1:33:27	.	.

NGL

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:06	0:04:55	0:00:06	0:14:59	0:00:48	0:40:28
3	0:00:31	0:19:00	0:00:31	1:07:48	0:05:42	3:14:13
4	.	0:25:20	.	1:30:16	.	.

We developed three different behaviours for this domain. Agent ALL picks every possible spanner on its way to the gate, while agent SME picks exactly as many spanners as are needed to tighten the nuts at the gate. Agent SGL takes a single spanner and rushes to the gate, and can then only tighten one nut. Our algorithm learnt the following formulas, that perfectly recognize agent ALL:

$$\forall x \in \text{Spanner}.\ \exists y \in \text{Operator}.\ \Diamond\Box\,\text{carrying}(y, x) \tag{11}$$

$$\forall x \in \text{Spanner}.\ \exists y \in \text{Location}.\ \text{at}(x, y) \wedge \Diamond\neg\text{at}(x, y) \tag{12}$$

Formula (11) expresses that every spanner will be picked up by the (only) operator and carried for the rest of the plan, and Formula (12) expresses that every spanner will be moved from its initial position.

When searching formulas with a single variable, we split the predicates so that the maximum arity of a fluent is 1. Our algorithm outputs the following very simple formula, which was learnt in a few seconds, while completely characterizing the behaviour of agent ALL:

$$\forall x \in \text{Spanner}.\ \Diamond\,\text{carrying}_2(x)$$

8 Conclusion

In this paper, we have presented a method to learn temporal logic formulas that recognize agents based on examples of their behaviours. We showed that such formulas can be learned using an algorithm that boils down to a reduction to MaxSAT, and that very few examples are sometimes enough to perfectly capture the behaviour of an agent on instances that can differ from the ones used in the training set. This justifies the cost of resorting to a first-order language, which generalizes to new instances, but is also very concise and easily readable by a human. The formulas that we learn can serve as higher-order descriptions of the behaviour of a planning agent.

In future works too, we wish to tailor our algorithm and our datasets so that they can generate domain-specific control knowledge. More specifically, we wish to work on its integration into various systems that can be guided with temporal logic, be them automated planning systems [2] or reinforcement learning agents [27]. This is in line with previous works on generalized planning, which learn logic-based policies fully capable of solving a set of planning problems, out of a set of example instances and plans [3, 9].

Acknowledgements

The author would like to thank Martin C. Cooper for his insightful advices and his careful proofreading of this manuscript. The author also wishes to thank the anonymous reviewers of ECAI 2024 for their remarks and suggestions.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(87)90052-6. URL https://www.sciencedirect.com/science/article/pii/0890540187900526.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116 (1):123–191, 2000. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00071-5. URL https://www.sciencedirect.com/science/article/pii/S0004370299000715.

[3] B. Bonet, G. Frances, and H. Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2703–2710, 2019.

[4] A. Camacho and S. A. McIlraith. Learning interpretable models expressed in linear temporal logic. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):621–630, May 2021. doi: 10.1609/icaps.v29i1.3529. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3529.

[5] T. de la Rosa and S. McIlraith. Learning domain control knowledge for TLPlan and beyond. In *Proceedings of the ICAPS-11 Workshop on Planning and Learning (PAL)*, 2011.

[6] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer Berlin Heidelberg, 2008.

[7] N. Fijalkow and G. Lagarde. The complexity of learning linear temporal formulas from examples. In J. Chandlee, R. Eyraud, J. Heinz, A. Jardine, and M. van Zaanen, editors, *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 237–250. PMLR, 23–27 Aug 2021. URL https://proceedings.mlr.press/v153/fijalkow21a.html.

[8] V. Fionda and G. Greco. The complexity of LTL on finite traces: Hard and easy fragments. In *AAAI Conference on Artificial Intelligence*, 2016. URL https://api.semanticscholar.org/CorpusID:31447386.

[9] G. Francès, B. Bonet, and H. Geffner. Learning general planning policies from small examples without supervision. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11801–11808. AAAI Press, 2021. doi: 10.1609/AAAI.V35I13.17402. URL https://doi.org/10.1609/aaai.v35i13.17402.

[10] J.-R. Gaglione, D. Neider, R. Roy, U. Topcu, and Z. Xu. Learning linear temporal properties from noisy data: a MaxSAT-based approach. In *Automated Technology for Verification and Analysis: 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18–22, 2021, Proceedings 19*, pages 74–90. Springer, 2021.

[11] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[12] S. Hölldobler and V.-H. Nguyen. An efficient encoding of the at-most-one constraint. *Technical Report. Technische Universitäat Dresden*, 2013.

[13] R. Horčík and D. Fišer. Endomorphisms of lifted planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 174–183, 2021.

[14] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9. URL https://doi.org/10.1007/978-1-4684-2001-2_9.

[15] J. Kim, C. Muise, A. J. Shah, S. Agarwal, and J. A. Shah. Bayesian inference of linear temporal logic specifications for contrastive explanations. International Joint Conference on Artificial Intelligence, 2019.

[16] D. E. Knuth. *The Art of Computer Programming*, volume 4A. Addison Wesley, 2020.

[17] A. Lequen. Implementation of an FTL-based classifier learner for PDDL planning, Aug. 2024. URL https://doi.org/10.5281/zenodo.13354031.

[18] A. Lequen. NP-hardness proof for the FTL learning problem, Aug. 2024. URL https://doi.org/10.5281/zenodo.13354790.

[19] N. Markey. Temporal logic with past is exponentially more succinct. *Bull. EATCS*, 79:122–128, 2003. URL https://api.semanticscholar.org/CorpusID:14823816.

[20] C. Mascle, N. Fijalkow, and G. Lagarde. Learning temporal formulas from examples is hard. *arXiv*, 2023.

[21] D. Neider and I. Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

[22] V.-H. Nguyen and S. T. Mai. A new method to encode the at-most-one constraint into SAT. In *Proceedings of the 6th International Symposium on Information and Communication Technology*, SoICT '15, page 46–53, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338431. doi: 10.1145/2833258.2833293. URL https://doi.org/10.1145/2833258.2833293.

[23] A. Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 46–57, 1977. URL https://api.semanticscholar.org/CorpusID:117103037.

[24] R. Raha, R. Roy, N. Fijalkow, and D. Neider. Scalable anytime algorithms for learning fragments of linear temporal logic. In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280, Cham, 2022. Springer International Publishing.

[25] H. Riener. Exact synthesis of LTL properties from traces. In *2019 Forum for Specification and Design Languages*, pages 1–6. IEEE, 2019. doi: 10.1109/FDL.2019.8876900. URL https://doi.org/10.1109/FDL.2019.8876900.

[26] M. Shvo, A. C. Li, R. T. Icarte, and S. A. McIlraith. Interpretable sequence classification via discrete optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9647–9656, 2021.

[27] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 452–461. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[28] F. Vaandrager, B. Garhewal, J. Rot, and T. Wißmann. A new approach for active automata learning based on apartness. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 223–243. Springer, 2022.

[29] C. Zhou, C. Sun, Z. Liu, and F. Lau. A C-LSTM neural network for text classification. *arXiv*, 11 2015.

Learning Interpretable Classifiers for PDDL Planning NP-hardness proof

Arnaud Lequen[a,*]

[a]IRIT, Université Toulouse III - Paul Sabatier
ORCID (Arnaud Lequen): https://orcid.org/0000-0003-0339-0967

Abstract. This document contains the proof of NP-hardness of the problem we tackle in the main article. In this document, we show that with our construction, a negative Set Cover instance results in a negative $\mathcal{L}_{\text{FTL}}$ learning instance. The rest of the proof is, in essence, the same as in the main article.

1 The $\mathcal{L}_{\text{FTL}}$ learning problem

1.1 Problem definition

In this section, we introduce the main problem we are tackling in this paper.

Score function Our problem takes in input a *score function*, which associated a score $\sigma(\langle t, \mathcal{I}\rangle) \in \mathbb{R}$ to each trace. We say that an instantiated trace $\langle t, \mathcal{I}\rangle$ is *positive* iff $\sigma(\langle t, \mathcal{I}\rangle) \geq 0$. Otherwise, the instantiated trace is said to be *negative*.

In the following, we use $[\langle t, \mathcal{I}\rangle \models \psi]$ as a shorthand for the function equal to 1 if $\langle t, \mathcal{I}\rangle \models \psi$ and equal to 0 otherwise. The score function generalizes to formulas $\psi \in \mathcal{L}_{\text{FTL}}$ as follows:

$$\sigma^{\mathsf{T}}(\psi) = \sum_{\langle t, \mathcal{I}\rangle \in \mathsf{T}} \sigma(\langle t, \mathcal{I}\rangle)\, [\langle t, \mathcal{I}\rangle \models \psi]$$

Problem 1. *$\mathcal{L}_{FTL}$ learning*
Input: $\mathcal{D}$ *a domain*
T a set of instantiated traces
$r \in \mathbb{N}$ the maximum number of logical operators in the output formula
$q \in \mathbb{N}$ the maximum number of quantifiers
$\sigma : \mathsf{T} \to \mathbb{R}$ a function called the score function
Output: *A formula $\psi \in \mathcal{L}_{FTL}$ such that ψ has at most r logical operators, and q quantifiers, and $\sum_{\langle t, \mathcal{I}\rangle \in \mathsf{T}} \sigma(\langle t, \mathcal{I}\rangle)\, [\langle t, \mathcal{I}\rangle \models \psi]$ is maximal*

1.2 Complexity

The decision problem associated to the $\mathcal{L}_{\text{FTL}}$ learning problem is the problem for which the output is *Yes* iff *there exists a formula $\psi \in \mathcal{L}_{FTL}$ satisfying the requirements above, and with score $\sigma^{\mathsf{T}}(\psi) \geq \ell$*, where ℓ is given in input.

* Email: Arnaud.Lequen@irit.fr

Proposition 1. *The decision problem associated to the $\mathcal{L}_{FTL}$ learning problem is NP-hard*

To show this result, we start by introducing the NP-hard problem we reduce to ours.

Problem 2. *Set Cover*
Input: *A set $U = \{1, \ldots, n\}$*
A set S of subsets: $S = \{S_1, \ldots, S_m\} \subseteq 2^U$
$k \in \mathbb{N}$
Output: *Yes* iff *there exists a subset $T \subseteq S$ such that $|T| \leq k$ and $\cup_{s \in T} s = U$*
No otherwise

Proof of Proposition 1. Using the notation above, let us consider an instance of *Set Cover*. We build an instance of *$\mathcal{L}_{FTL}$ learning* that is positive (i.e. outputs *Yes*) *iff* the *Set Cover* instance is positive.

The proof consists in showing that a set of positive traces can be described by a formula $\mathcal{L}_{\text{FTL}}$ satisfying the constraints in input iff there exists a set cover of size at most k. Each of the positive traces is associated to an element j of U, and contains a single state (and thus, temporal modalities have no effect). This single state carries the information as to which subsets contain j. The information consists of fluents of the form $\text{in}(S_i)$. For instance, if the j-th trace is $\langle\{\text{in}(S_1), \text{in}(S_3)\}\rangle$, then only sets $S_1, S_3 \in S$ contain element $j \in U$.

Each subset S_i of S, in the Set Cover instance given in input, is associated a unique type Set_i in the PDDL domain we build. As our $\mathcal{L}_{\text{FTL}}$ formulas quantify on these types, and since we restrict the number of quantifiers to $q = k$, any output formula of the $\mathcal{L}_{\text{FTL}}$ learning problem can only reason on at most k subsets among those in S. If k quantifiers are enough to distinguish between the positive traces (associated to elements $j \in U$) and a mock trace, then k subsets are enough to cover all elements of U. Otherwise, no set cover of size at most k exists.

More formally, let us define the input of our *$\mathcal{L}_{FTL}$ learning* instance as follows: the shared domain is $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}, \mathcal{T}\rangle$, where the set of types is $\mathcal{T} = \{\text{Set}_1, \ldots, \text{Set}_m, \text{Set}\}$, $\mathcal{P} = \{\text{in}\}$ (with $ar(\text{in}) = 1$ and $\tau_{\text{in}}(1) = \text{Set}$), and $\mathcal{A} = \varnothing$ (since all traces have length 1, no action is needed).

Each instantiated trace is associated its own instance, which is in turn associated to a unique element $j \in U$: for $j \leq n$, $\mathcal{I}_j = \langle \mathcal{O}, \mathcal{H}, I_j, G_j\rangle$. The shared set of objects contains the sets of the Set Cover input instance, and is such that $\mathcal{O} = \{S_1, \ldots, S_m, d\}$ (where

d is a mock object that we use to discriminate traces later). The type hierarchy $\mathcal{H} = \{\{S_i\} \mid S_i \in S\} \cup \mathcal{O}$ associates each set to its own unique type in our PDDL domain: types are such that $\tau(S_i) = \text{Set}_i$ for all $i \leq m$, and $\tau(\mathcal{O}) = \text{Set}$. The information regarding which sets j belongs to is encoded in the initial state and goal of $\mathcal{I}_j$: for $j \leq n$, $I_j = G_j = \{\text{in}(S_i) \mid j \in S_i\}$.

We also define a mock, negative instance $\mathcal{I}_d = \langle \mathcal{O}, \mathcal{H}, I_d, G_d \rangle$, with $I_d = \{\text{in}(d)\}$, and $G_d = I_d$. We have $\mathcal{T} = \{\langle t_j, \mathcal{I}_j \rangle \mid j \leq n\} \cup \{\langle t_d, \mathcal{I}_d \rangle\}$, where $t_j = \langle I_j \rangle$ and $t_d = \langle I_d \rangle$. We set $r = m - 1$, $q = k$, and the score $\sigma(\langle t_j, \mathcal{I}_j \rangle)$ of every trace to 1, except $\langle t_d, \mathcal{I}_d \rangle$ for which $\sigma(\langle t_d, \mathcal{I}_d \rangle) = -1$. The decision problem consists in finding whether there exists a formula $\psi \in \mathcal{L}_{\text{FTL}}$ that can that is satisfied by every trace, except $\langle t_d, \mathcal{I}_d \rangle$. Thus, the score threshold we set is $\ell = m$. Let us now show that this instance is positive iff our *Set Cover* instance is positive.

Suppose that there exists a set cover $T = \{S_{i_1}, \ldots, S_{i_k}\}$. Then there exists a formula $\psi \in \mathcal{L}_{\text{FTL}}$ with score exactly m:

$$\psi := \exists x_1 \in \text{Set}_{i_1}, \ldots, \exists x_k \in \text{Set}_{i_k}. \bigvee_{j \leq k} \text{in}(x_j)$$

We have that $\langle t_d, \mathcal{I}_d \rangle \not\models \psi$, but all other traces satisfy ψ, since T is a set cover. So the formula has score exactly m, and is a positive instance of $\mathcal{L}_{FTL}$ *learning*.

Now suppose that there exists no set cover of size at most k for the input instance. By contradiction, suppose that there exists a formula ψ with score m (which is the maximum possible score). We denote T' the set of types on which ψ quantifies upon. We have that $|T'| \leq k$.

Since there exists no set cover of size at most k, then there exists $w \leq n$ such that $w \notin \cup_{s \in T'} s$. We proceed to show that if $\langle t_w, \mathcal{I}_w \rangle \models \psi$, then $\langle t_d, \mathcal{I}_d \rangle \models \psi$, which contradicts the fact that ψ has score m.

Since all instances share the same set of objects (and the same environments), in the rest of this proof, we denote $t, e \models \psi$ as a shorthand for $\langle t, \mathcal{I} \rangle, e \models \psi$. We introduce two lemmas, which are stronger results than needed here, but that are necessary for the proof. Let $C_w = \{o \in \mathcal{O} \mid \text{in}(o) \in I_w\}$, $C_d = \{d\}$, $\overline{C_w} = \mathcal{O} \setminus C_w$, and $\overline{C_d} = \mathcal{O} \setminus C_d$.

Lemma 2. *Let $\psi \in \mathcal{L}_{FTL}$, $v \in \{d, w\}$, and $v' \in \{d, w\} \setminus \{v\}$. If there exists an environment e s.t. $t_v, e \models \psi$, then there exists an environment e' s.t. $t_{v'}, e' \models \psi$, and e' is such that for every $x \in \mathcal{X}$ s.t. $x[e] \in C_v$ (resp. $\overline{C_v}$), we have $x[e'] \in C_{v'}$ (resp. $\overline{C_{v'}}$)*

In the following lemma, for $v \in \{d, w\}$, we say that two environments e and e' are *indistinguishable for C_v* if, for every $x \in \mathcal{X}$, $x[e] \in C_v$ iff $x[e'] \in C_v$ (and thus, $x[e] \in \overline{C_v}$ iff $x[e'] \in \overline{C_v}$).

Lemma 3. *Let $\psi \in \mathcal{L}_{FTL}$, $v \in \{d, w\}$. Suppose that there exists an environment e s.t. $t_v, e \models \psi$. Then for every environment e' such that e and e' are indistinguishable for C_v, we have $t_v, e' \models \psi$.*

Intuitively, these lemmas translate the fact that trace t_w (resp. t_d) can not distinguish two objects that occur in I_w (resp. I_d), and that objects of C_w (resp. $\overline{C_w}$) that occur in an environment e that satisfies ψ with t_w can be replaced by objects of C_d (resp. $\overline{C_d}$) to create a new environment e', so that ψ is satisfied by t_d with e'.

We prove both lemmas simultaneously, by induction over the structure of ψ. We only show the cases where $v = w$, although the cases where $v = d$ are shown analogously. We start by proving by induction that these lemmas hold when $\psi = \varphi \in \mathcal{L}_{\text{TL}}$.

The case $\varphi = \text{in}(x)$ is immediate, by construction.

Suppose that $\varphi = \neg\varphi'$. (Lemma 2) Suppose that there exists e s.t. $t_w, e \models \varphi$. Then $t_w, e \not\models \varphi'$, and there exists e' such that $t_d, e' \not\models \varphi'$ (which is something that we obtain by using the induction hypothesis of both lemmas). So $t_d, e' \models \varphi$. The converse is shown in a similar way. (Lemma 3) Suppose that $t_w, e \models \varphi$. Then $t_w, e \not\models \varphi'$, and for every environment e' such that e and e' are indistinguishable for C_v, $t_w, e' \not\models \varphi'$ (by induction hypothesis). So $t_w, e' \models \varphi$.

Suppose that $\varphi = \varphi_1 \wedge \varphi_2$. (Lemma 2) Suppose that there exists e such that $t_w, e \models \varphi$. By induction hypothesis, there exists e'_1 and e'_2 such that $t_d, e'_1 \models \varphi_1$ and $t_d, e'_2 \models \varphi_2$. Using the induction hypothesis of Lemma 3, we have that (since e'_1 and e'_2 are indistinguishable for C_d) there exists e' an environment such that $t_d, e' \models \varphi_1$ and $t_d, e' \models \varphi_2$, so $t_d, e' \models \varphi$. Showing that the induction hypothesis of Lemma 3 carries over is straightforward.

The cases of temporal modalities are immediate, since all traces have length 1.

Suppose that $\psi = \exists x \in E.\psi'$, where $E \in \mathcal{T}$. Now suppose that $E = \text{Set}_i$ for some i (the proof when $E = \text{Set}$ is similar). (Lemma 2) Suppose that there exists e such that $t_w, e \models \psi$. Then there exists $o \in \mathcal{O}$, $\tau(o) = \text{Set}_i$ such that $t_w, e[x := o] \models \psi$. Note that $o \in \overline{C_w}$ (by hypothesis on t_w, since w is not covered by the set of subsets T' on which ψ quantifies upon). There exists $o' \in \overline{C_d}$ and an environment e' such that $t_w, e'[x := o'] \models \psi'$. Since $o \in \overline{C_d}$, Lemma 3 ensures that $t_w, e'[x := o] \models \psi$. So $t_w, e' \models \psi$. Lemma 3 is proven in a similar way, since by induction hypothesis, any environment e'' indistinguishable with e' for C_d.

Suppose that $\psi = \forall x \in E.\psi'$, where $E \in \mathcal{T}$. Since there exists a unique $o \in \mathcal{O}$ such that $\tau(o) = \text{Set}_i$ for all i, the case $E = \text{Set}_i$ can be shown in the same way as above. Now suppose that $E = \text{Set}$, and that there exists an environment e such that $t_w, e \models \psi$. For all $o \in \mathcal{O}$, $t_w, e[x := o] \models \psi'$. Let $o_1 \in C_w$ and $o_2 \in \overline{C_w}$. Then by induction hypothesis, there exist e'_1, e'_2 two environments, $o'_1 \in C_d$ and $o'_2 \in \overline{C_d}$ such that $t_d, e'_1[x := o'_1] \models \psi'$ and $t_d, e'_2[x := o'_2] \models \psi'$. Since e'_1 and e'_2 are indistinguishable for C_d, and by induction hypothesis, we have that $t_d, e'_1[x := o'_1] \models \psi'$ and $t_d, e'_1[x := o'_2] \models \psi'$. Since, for any $o''_1 \in C_d$ (and $o''_2 \in \overline{C_d}$), $e'_1[x := o'_1]$ and $e'_1[x := o''_1]$ are indistinguishable for C_d (as well as $e'_1[x := o'_2]$ and $e'_1[x := o''_2]$), we have that, for any $o'' \in \mathcal{O}$, $t_d, e'_1[x := o''] \models \psi'$. Hence, $t_d, e'_1 \models \psi'$. Since any other environment e'' indistinguishable with e'_1 over C_d could have been chosen, Lemma 3 also holds.

This concludes the proof of both lemmas. An immediate corollary of Lemma 2 is that $\langle t_w, \mathcal{I}_w \rangle \models \psi$ iff $\langle t_d, \mathcal{I}_d \rangle \models \psi$. As a consequence, $\sigma^{\mathcal{T}}(\psi) < m$, which is a contradiction with ψ being a solution to our $\mathcal{L}_{\text{FTL}}$ learning instance.

As such, the $\mathcal{L}_{\text{FTL}}$ learning instance we built is positive iff the set cover instance is positive. As a consequence, the $\mathcal{L}_{\text{FTL}}$ learning problem is NP-hard. $\square$

Learning Interpretable Classifiers for PDDL Planning Experimental results

Arnaud Lequen[a,*]

[a]IRIT, Université Toulouse III - Paul Sabatier
ORCID (Arnaud Lequen): https://orcid.org/0000-0003-0339-0967

Abstract. This document contains additional experimental results, in the form of an additional dataset (domain Rovers), more detailed statistics, and additional example formulas.

1 Example of learnt formulas

Childsnack Problem Childsnack consists in making sandwiches and serving them to a group of children, some of whom are allergic to gluten. Sandwiches can only be prepared in the kitchen, and then have to be put on trays, which is the only way they can be brought to the children for service.

We designed three different agents that solve Childsnack instances. Agents NGF and NGL compute solution plans of minimal size, and differ in that agent NGF makes sandwiches with *no gluten first*, and agent NGL makes sandwiches with *no gluten last*. Both agents make all sandwiches, put them on a tray, then serve the children. Agent GS greedily serves children: as soon as a sandwich is made, it is put on a tray and brought to a child. It also prioritizes gluten-free sandwiches.

As shown on Table 4, we manage to learn formulas that perfectly capture the test set, even when the number of examples is very limited, and when few quantifiers and logical operators are allowed. The following formula, which expresses that every sandwich is put on a tray right after it is prepared, perfectly captures the behaviour of agent GS, compared to the other agents:

$$\forall x \in \text{Sandwich}.\, \exists y \in \text{Tray}.\, \text{notprepared}(x) \,\mathsf{U}\, \bigcirc\, \text{on}(x, y)$$

Other formulas include the following:

$$\forall x \in \text{Kitchen}.\, \exists y \in \text{Tray}.\, \Diamond(\text{at}(y, x) \wedge \overline{\Diamond}\neg\text{at}(y, x)) \tag{1}$$

$$\forall x \in \text{Kitchen}.\, \exists y \in \text{Tray}.\, \Diamond(\neg\text{at}(y, x) \wedge \bigcirc\text{at}(y, x)) \tag{2}$$

Formula (1) expresses that agent GS eventually comes back to the (only) kitchen with some tray y, even though the tray was brought out of the kitchen at some point in the past. Formula (2) expresses the same idea, but pinpoints the moment when a tray is brought back to the kitchen. The formula perfectly captures our test set, but does not perfectly capture our *training* set. Indeed, the smallest instance of our training set contains as many children as there are trays, and thus, no tray has to be brought back to the kitchen. Our use of a reduction to MaxSAT allows us to be resilient to this kind of edge cases, and the formula above is satisfactory despite not perfectly fitting the training set.

* Email: Arnaud.Lequen@irit.fr

When it comes to the other agents, the following formula perfectly captures the behaviour of agent NGF. It does not capture the behaviour of agent NGL since it starts with a gluten-free sandwich, and it does not capture the behaviour of agent GS since the sandwich is not immediately shipped.

$$\exists x \in \text{Sandwich}.\, \bigcirc(\text{no\_gluten\_sandwich}(x) \wedge \bigcirc\, \text{at\_kitchen\_sandwich}(x))$$

The behaviour of agent NGL, which is the only one to start with making sandwiches with gluten, is simply captured by the following formula:

$$\exists x \in \text{Sandwich}.\, \bigcirc(\text{at\_kitchen\_sandwich}(x) \wedge \neg\text{no\_gluten\_sandwich}(x))$$

Spanner Instances of the Spanner domain involve an operator that has to go from a shed to a gate to tighten some nuts, passing through a sequence of locations where single-use spanners can be picked up. Once a location is left, it can not be returned to. Thus, collecting enough spanners before reaching the gate is essential.

We developed three different behaviours for this domain. Agent ALL picks every possible spanner on its way to the gate, while agent SME picks exactly as many spanners as are needed to tighten the nuts at the gate. Agent SGL takes a single spanner and rushes to the gate, and can then only tighten one nut. Our algorithm learnt the following formulas, that perfectly recognize agent ALL:

$$\forall x \in \text{Spanner}.\, \exists y \in \text{Operator}.\, \Diamond\Box\, \text{carrying}(y, x) \tag{3}$$

$$\forall x \in \text{Spanner}.\, \exists y \in \text{Location}.\, \text{at}(x, y) \wedge \Diamond\neg\text{at}(x, y) \tag{4}$$

Formula (3) expresses that every spanner will be picked up by the (only) operator and carried for the rest of the plan, and Formula (4) expresses that every spanner will be moved from its initial position.

When searching formulas with a single variable, we split the predicates so that the maximum arity of a fluent is 1. Our algorithm outputs the following very simple formulas, which were learnt in a few seconds, while completely characterizing the behavior of agent ALL:

$$\forall x \in \text{Spanner}.\, \Diamond\, \text{carrying}_2(x)$$

$$\forall x \in \text{Spanner}.\, \text{useable}(x) \,\mathsf{U}\, \text{carrying}_2(x)$$

The behaviour of agent SGL, which is the only one who fails to solve the problem, is captured by the following formula, for instance:

$$\forall x \in \text{Nut}.\, \Box\, \text{loose}(x)$$

However, no formula could capture the behaviour of agent SME in a significant way.

Rovers Domain Rovers simulates a planetary exploration mission, where a fleet of mobile rovers has to navigate between various waypoints on a planet to collect data or samples, and to transmit the data back to a lander. The rovers have instruments, which have to be calibrated before they can collect data.

The set of instances we designed all required a single rover to collect two kinds of samples (soil and rock), and to take pictures of one or various objectives. We designed three different agents, that each prioritize a certain kind of tasks above all others. Agent CIF will calibrate its cameras first and take a picture of the objectives, agent CRF will collect rock samples first, and agent CSL will collect soil samples first.

The following formula captures the behaviour of agent CIF with 93.3% accuracy. It translates the fact that CIF is the only Rovers agent to not fetch any sample at first. However, since not all samples are within reach of three actions in all instances, other agents may also satisfy this property on some particular instances.

$$\forall x \in \text{Store}.\ \bigcirc \bigcirc \bigcirc\, \text{empty}(x)$$

A property that describes agent CIF's behaviour with 91.7% accuracy is the following. It expressed the fact that, within two actions, agent CIF has calibrated its camera (on at least one rover).

$$\exists x \in \text{Rover}.\ \exists y \in \text{Camera}.\ \bigcirc(\text{calibrated}(x, y) \vee \bigcirc\, \text{calibrated}(x, y))$$

The following formula captures agent CRF's behaviour with 81.7% accuracy (where Goal(communicated_soil_data(x)) means that communicated_soil_data(x) is in the goal). It expresses the fact that soil samples are not collected until some rock sample is processed. Even though this formula often successfully discriminates agents CRF and CSF, it sometimes with to tell agent CRF from agent CIF.

$$\begin{aligned}&\forall x \in \text{Waypoint}.\ \exists y \in \text{Waypoint}.\\ &\text{Goal}(\text{communicated\_soil\_data}(x)) \Rightarrow\\ &(\text{at\_soil\_sample}(x)\ \mathsf{U}\ \text{communicated\_rock\_data}(y))\end{aligned}$$

2 Additional statistics

For each domain and for each domain-specific behaviour, the total computation time and the total number of formulas found are depicted in Table 1 and Table 5, respectively. In addition, Table 4 presents the best accuracy obtained by a single formula for each configuration.

Figures 1, 2 and 3 summarize the distributions of the accuracies of the formulas we learnt.

Table 1: Total computation times for Childsnack (hh:mm:ss). Rows correspond to the number of allowed logical connectors. Dots correspond to configuration where the run did not terminate.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:08	0:05:03	0:00:08	0:16:43	0:01:14	0:44:48
3	0:00:35	0:19:53	0:00:35	1:07:27	0:06:46	3:35:18
4	.	0:25:27	.	1:32:07	.	.

NGF

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:08	0:05:08	0:00:08	0:17:07	0:01:09	0:47:51
3	0:00:37	0:20:13	0:00:37	1:11:13	0:07:05	3:44:11
4	.	0:26:29	.	1:33:27	.	.

NGL

# ins.	1		2		3	
q	1	2	1	2	1	2
2	0:00:06	0:04:55	0:00:06	0:14:59	0:00:48	0:40:28
3	0:00:31	0:19:00	0:00:31	1:07:48	0:05:42	3:14:13
4	.	0:25:20	.	1:30:16	.	.

Table 2: Total computation times for Spanner (hh:mm:ss). Rows correspond to the number of allowed logical connectors. Dots correspond to configuration where the run did not terminate.

ALL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:03	0:07:11	0:00:03	0:31:13	0:00:37	1:28:33
3	0:00:12	0:25:32	0:00:12	2:08:30	0:02:58	6:35:33
4	.	0:32:00	.	.	.	.

SGL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:03	0:07:13	0:00:03	0:30:12	0:00:35	1:27:01
3	0:00:12	0:25:30	0:00:12	1:58:35	0:02:17	6:01:47
4	.	0:31:45	.	2:51:37	.	.

SME

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:03	0:07:14	0:00:03	0:30:59	0:00:37	1:28:48
3	0:00:13	0:25:24	0:00:13	2:03:55	0:02:48	6:21:49
4	.	0:31:41	.	2:58:58	.	.

Table 3: Total computation times for Rovers (hh:mm:ss). Rows correspond to the number of allowed logical connectors. Dots correspond to configuration where the run did not terminate.

CIF

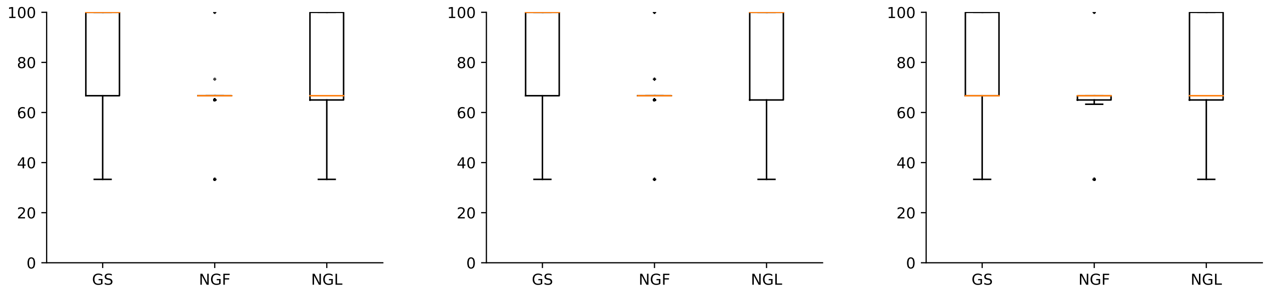
# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:42	0:26:35	0:00:42	1:14:41	0:05:23	3:39:21
3	0:03:44	2:20:57	0:03:44	5:02:09	0:51:29	.
4	.	.	.	.	.	.

CRF

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:39	0:25:41	0:00:39	1:11:16	0:05:27	3:28:22
3	0:03:06	1:50:59	0:03:06	5:47:15	0:29:34	.
4	.	.	.	.	.	.

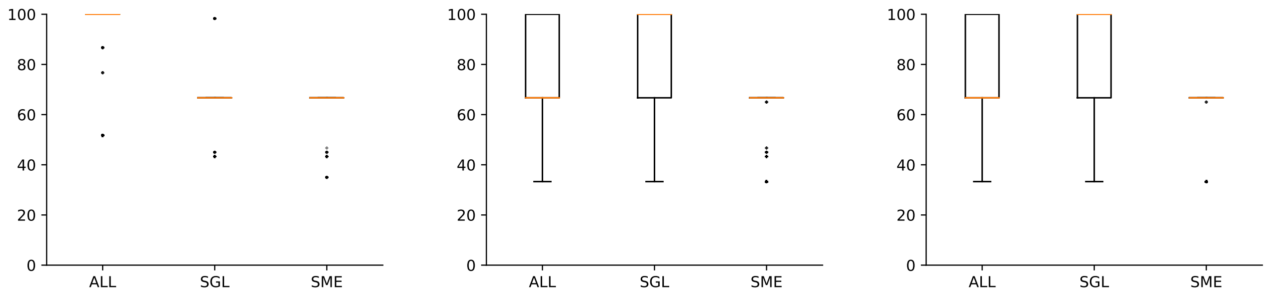
CSF

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0:00:41	0:25:12	0:00:41	1:03:39	0:05:01	3:06:28
3	0:03:41	1:58:30	0:03:41	4:05:01	0:35:34	.
4	.	2:18:24	.	.	.	.



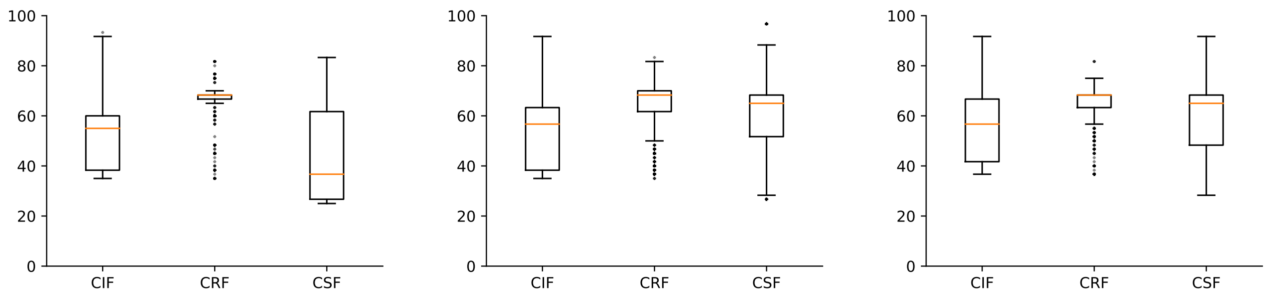
(a) 1 instance (b) 2 instances (c) 3 instances

Figure 1: Distribution of the percentages of accuracy on the test set of the formulas learnt for Childsnack problems. Each subfigures gives the distributions for each agent, along with the number of instances used in the training set.



(a) 1 instance (b) 2 instances (c) 3 instances

Figure 2: Distribution of the percentages of accuracy on the test set of the formulas learnt for Spanner problems. Each subfigures gives the distributions for each agent, along with the number of instances used in the training set.



(a) 1 instance (b) 2 instances (c) 3 instances

Figure 3: Distribution of the percentages of accuracy on the test set of the formulas learnt for Rovers problems. Each subfigures gives the distributions for each agent, along with the number of instances used in the training set.

Table 4: Proportion of traces correctly classified by the most accurate formula on our test set, expressed in percentage (%). Each table of the first, second, and third row presents a different agent solving a set of Childsnack, Spanner, and Rovers problems, respectively. *# ins.* indicates the number of instances in the training set, q the number of quantifiers allowed, and $|\varphi|$ the number of logical operators allowed. Dots correspond to configuration where the run did not terminate.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
$\|\varphi\| = 2$	67	100	67	100	67	100
3	100	100	100	100	100	100
4	.	100	.	100	.	.

NGF

1		2		3	
1	2	1	2	1	2
67	67	67	67	67	67
100	100	100	100	100	67
.	.	.	.	.	.

NGL

1		2		3	
1	2	1	2	1	2
100	100	100	100	100	100
100	100	100	100	100	100
.	.	.	.	.	.

ALL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\|\varphi\| = 2$	.	100	.	100	67	100
3	.	100	.	100	67	100
4	.	100	.	.	.	.

SGL

1		2		3	
1	2	1	2	1	2
.	67	.	100	100	100
.	67	.	100	100	100
.	98	.	100	.	.

SME

1		2		3	
1	2	1	2	1	2
.	67	.	67	67	67
.	67	.	67	67	67
.	67	.	67	.	.

CIF

# ins.	1		2		3	
q	1	2	1	2	1	2
$\|\varphi\| = 2$	92	78	92	78	92	78
3	93	.	93	90	92	.
4	.	.	.	.	.	.

CRF

1		2		3	
1	2	1	2	1	2
75	82	75	82	75	82
75	77	75	82	75	.
.	.	.	.	.	.

CSF

1		2		3	
1	2	1	2	1	2
42	83	42	83	82	92
62	.	62	97	92	.
.	.	.	.	.	.

Table 5: Total number of formulas found for each agent, and for each configuration. Each table of the first, second, and third row presents a different agent solving a set of Childsnack, Spanner, and Rovers problems, respectively.

GS

# ins.	1		2		3	
q	1	2	1	2	1	2
$\|\varphi\| = 2$	22	110	22	112	22	123
3	92	624	92	624	92	671
4	.	746	.	757	.	.

NGF

1		2		3	
1	2	1	2	1	2
23	126	23	126	23	140
92	670	92	679	94	723
.	797	.	858	.	.

NGL

1		2		3	
1	2	1	2	1	2
7	76	7	78	8	92
79	604	79	681	82	725
.	784	.	845	.	.

ALL

# ins.	1		2		3	
q	1	2	1	2	1	2
$\|\varphi\| = 2$	.	71	.	257	20	257
3	.	377	.	1043	78	1040
4	.	490	.	.	.	.

SGL

1		2		3	
1	2	1	2	1	2
.	62	.	226	20	225
.	237	.	864	57	866
.	379	.	1160	.	.

SME

1		2		3	
1	2	1	2	1	2
.	59	.	246	20	246
.	237	.	898	78	902
.	379	.	1201	.	.

CIF

# i.	1		2		3	
q	1	2	1	2	1	2
2	43	572	43	740	79	800
3	200	1797	200	1200	200	.
4	.	.	.	.	.	.

CRF

1		2		3	
1	2	1	2	1	2
35	545	35	728	80	800
179	1595	179	1850	200	.
.	.	.	.	.	.

CSF

1		2		3	
1	2	1	2	1	2
43	511	43	725	80	800
200	1797	200	1850	200	.
.	1680	.	.	.	.