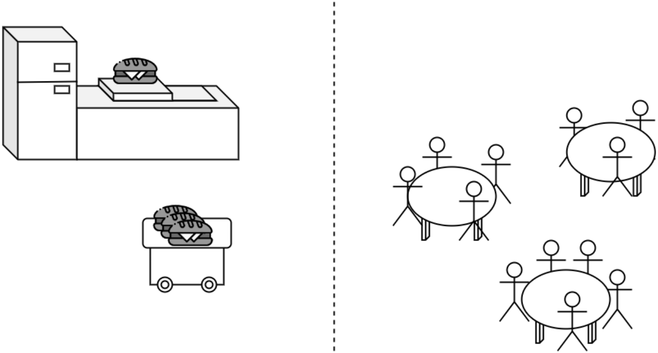
Learning Interpretable Classifiers for PDDL Planning

Arnaud Lequen



Problem: Serve sandwiches to children seated in a dining hall. Some children are allergic to gluten. Adequate sandwiches must be prepared in the kitchen, put on a tray, and brought to the tables

Properties of the **plans** (= sequences of actions) for the problem?

Multiple motives to reason about properties of plans:

- Landmarks, goal-achievement... characterize solution-plans
- Falsified invariants characterize partial plans that can not be extended into a solution (dead ends)

Learning such properties automatically is interesting in itself, and can extend beyond these applications

- Guiding search for a solution on bigger instances
- Behaviour recognition
- Reward function learning, goal recognition
- High-level policy description/explanation

In which unifying language to express these properties?

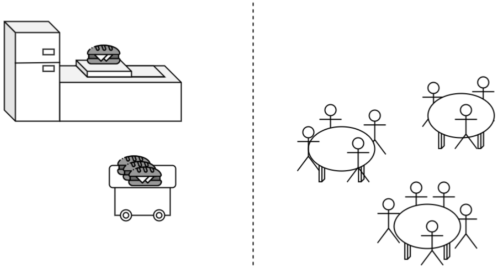
A language for expressing properties on traces

Linear Temporal Logic can express properties on sequences of states

- We extend it with first-order quantifications, so that
  - properties **generalize** to multiple instances
  - it takes advantage of the **PDDL planning model**
- We can then **learn a wide range of properties** that characterize a set of **example traces given in input**
- The language is very interpretable, even for non-experts

Language we use: First-Order Temporal Logic ($\mathcal{L}_{\mathsf{FTL}}$)

Childsnack in **typed PDDL**



*All children will **eventually** $\Diamond$ be served a sandwich*

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

*Children **always** $\Box$ sit at the same table*

$$\forall x \in \text{Child}.\ \exists y \in \text{Table}.\ \Box\, \text{sitting\_at}(x, y)$$

We can learn properties that characterize agents:

On the state that's ***next*** $\bigcirc$ *to the* ***next*** $\bigcirc$ *state, there exists a sandwich which is gluten-free and on some tray.*

$$\exists x \in \mathrm{Sandwich}.\, \exists y \in \mathrm{Tray}.\ \bigcirc \bigcirc(\mathrm{ontray}(x, y) \wedge \mathrm{no\_gluten\_sandwhich}(x))$$

Every sandwich is, at first, not prepared, ***until*** *U we reach a state where, in the* ***next*** $\bigcirc$ *state, it is on some tray:*

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

Model-checking example

Example with sandwich w_1 and tray r_1, for some plan π and trace t

State seq. t	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
on(w_1, r_1)	x	x	x	x	x	x	✓	✓	✓	✓	x
$\bigcirc$ on(w_1, r_1)	x	x	x	x	x	✓	✓	✓	✓	x	x
notprepared(w_1)	✓	✓	✓	✓	✓	x	x	x	x	x	x
φ	✓	✓	✓	✓	✓	✓	✓	✓	✓	x	x

where $\varphi :=$ notprepared(w_1) U $\bigcirc$ on(w_1, r_1)

Since φ is satisfied in s_0, we have that $t \models \varphi$

PDDL planning domain: $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{P}$ is a set of predicates, $\mathcal{A}$ a set of action schemas, and $\mathcal{T}$ a type tree.

We define our language $\mathcal{L}_{\mathsf{FTL}}$ such that:

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

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

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

where $x, x_1, \ldots, x_{ar(p)}$ are variables of $\mathcal{X}$ (a set of variables symbols), p a predicate of $\mathcal{P}$, and τ a type of $\mathcal{T}$.

Goal: Learn $\mathcal{L}_{\mathsf{FTL}}$ properties characterizing an agent A's behaviour

Outline of our method

- Build a set of positive and negative examples
  - Create planning instances and ask agents to solve them
  - Positive ex.: traces associated to plans made by agent A
  - Negative ex.: traces associated to plans made by other agents
- Assign a score $\sigma(t)$ in $[-100, 100]$ to each trace t
  - Positive traces t have positive scores $\sigma(t) > 0$ (*reward*)
  - Negative traces t have negative scores $\sigma(t) < 0$ (*penalty*)
- Find a formula $\psi \in \mathcal{L}_{\mathsf{FTL}}$ that gets the highest score:
  - ψ scores the points of t iff $t \models \psi$

Instantiated trace: pair $\langle t, \mathcal{I} \rangle$, where

t is a trace (= sequence of states associated to a plan)

$\mathcal{I}$ is a PDDL instance

Problem: $\mathcal{L}_{\mathsf{FTL}}$ learning

Input: $\mathcal{D}$ a PDDL domain

T a set of instantiated traces

$\sigma : T \to \mathbb{R}$ a function called the *score function*

$r \in \mathbb{N}$ the max number of logical operators

$q \in \mathbb{N}$ the max number of quantifiers

Output: A formula $\psi \in \mathcal{L}_{\mathsf{FTL}}$ such that ψ has at most r logical operators, and q quantifiers, and

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

is maximal

Associated decision problem

Is there a formula $\psi \in \mathcal{L}_{\mathsf{FTL}}$ that has score at least k?

Hardness

The $\mathcal{L}_{\mathsf{FTL}}$ learning problem is **NP-hard**

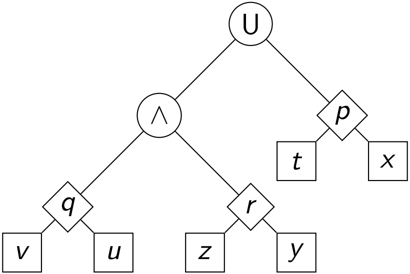
Upper bound

The $\mathcal{L}_{\mathsf{FTL}}$ learning problem is in **PSPACE**

TL Chains

$\mathcal{L}_{\mathrm{TL}}$ formulas can be represented using TL chains:

- ○ represent logical connectors
- ◇ represent predicates
- □ represent variables



Represents the quantifer-free FTL formula:

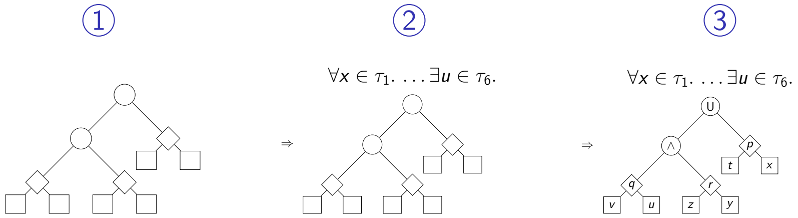
$$(q(v,u) \wedge r(z,y)) \, \mathrm{U} \, p(t,x)$$

Learning algorithm *(Simplified)*

① For all possible TL chains

② For all combinations of quantifiers and types

③ Call a **MaxSAT solver** to fill in the blanks in the chain, so that the formula fits best the dataset



Core constraints

Syntactic, semantic constraints: ensure well-formed formulas and consistency with the example traces

Very reminiscent of the model-checking algorithm for modal logic

Formula quality enhancement

Syntactic redundancies prevention: prevent formulas which have idempotent or involutive operators chained.
Ex: $\Diamond\Diamond\varphi$, $\neg\neg\varphi$, etc.

Variable visibility enforcement: ensure that all variables quantified upon are visible in the quantifier-free parts of the formula

MaxSAT is a cornerstone of our approach:

Learning problem: Maximize the score of the learnt formula

For each trace t, use a variable c_t

- c_t is true *iff* $c_t \models \psi$
- Soft clause: c_t with weight $\sigma(t)$

Score of the MaxSAT encoding = Score of the learnt formula ψ

Pros (compared to a pure SAT approach)

- Resilience to noise in the training set
- When no perfect formula exists, still learns a classifier with the best accuracy

To have distinctive behaviours, we hand-coded 3 domain-specific agents, for each of 3 different domains found in the IPC (Childsnack, Spanner, Rovers).

Training set

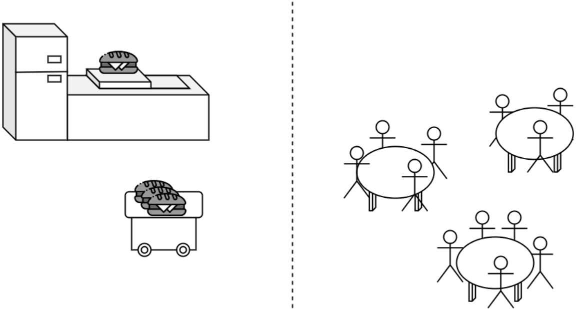
For each domain:

- Three small planning problems, and the plans of the agents

Test set

For each domain:

- 20 planning problems of various sizes, and the plans generated by the agents



Agents that solve Childsnack problems:

- NGL - optimally solve the problem, start with sandwiches with **gluten first**
- NGF - same as NGF but **start with gluten-free sandwiches**
- GS - **greedily** serves children: prepare a sandwich, put it on the tray, go and serve it directly. **Prioritizes gluten-free** sandwiches

Agents that solve Childsnack problems:

- NGL - optimally solve the problem, start with sandwiches with **gluten first**
- NGF - same as NGF but **start with gluten-free sandwiches**
- GS - **greedily** serves children: prepare a sandwich, put it on the tray, go and serve it directly. **Prioritizes gluten-free** sandwiches

They are respectively completely recognized by the following:

$$\forall x \in \text{Sandwich}.\ \bigcirc\ \neg \text{no\_gluten\_sandwich}(x)$$

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

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

Average time to find a (possibly imperfect) formula for a Childsnack agent, in seconds

Rows: # of allowed logical connectors. q: # of quantifiers
 # ins.: # of instances in the training set

# ins.	1		2		3	
q	1	2	1	2	1	2
$\lvert\varphi\rvert = 2$	0.4	2.9	0.4	9.3	3.6	22.5
3	0.4	1.9	0.4	6.2	4.4	17.9
4	.	2.0	.	6.7	.	.

Total computation times for Childsnack (top, hh:mm:ss), and total number of formulas (bottom), for agent GS.

Rows: # of allowed logical connectors. q: # of quantifiers
 # ins.: # of instances in the training set

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

# 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	.	.

Computation time

- Total computation times are shown, but lots of possibilities of parallelization
- A perfect solution is found within a couple minutes, but we let the algorithm run nevertheless

Few-shot learning

- Often, very few instances (even sometimes 1) are enough to learn a classifier that has the perfect score on our test set

MaxSAT for imperfect datasets

- Sometimes, the training set is so small that it contains a little noise
- The solver still learns imperfect formulas, that have high accuracy on the test set

Perspective

In this presentation

- A framework for learning interpretable formulas that generalize and characterize a set of plans
- Application to behaviour classification
- A reduction to MaxSAT that learns classifiers from small datasets in reasonable time

Perspective: Recombining formulas

- Combine the (sometimes imperfect) formulas into another more robust classifier

Perspective: Usage in search

- Assist a RL agent in its learning phase ($\approx$ Imitation learning)