Chapter 3

Implementation

For the moment, we only have implemented the *tic-tac-toe* and the *Field Field Kono* in python by using `pygame` (python library for video games).

The idea would be to learn from the actions performed by a player to train our system by using ILASP. Then we could make him play against this trained system and continue to train it that way.

3.1 First Method: maximize the chances of winning

3.1.1 Learning Exception Structure

Supposing we are at time $T = t$, we want the computer to induce a set of rules H such that:

- There is a move m such that for every answer set A of $B \cup H$, $does(player1, m, t) \in A$
- For every answer set A of $B \cup H$, for all move m_1 different from m, we have: $does(player1, m_1, t) \notin A$
- H also tries to take into account the possible future moves.

To reduce the hypothesis space, it would be interesting to focus only on *exception structures*, which means we want H to be of this form:

$$H = \left\{ \begin{array}{c} \texttt{does(player1, move\_p1\_t0\_1, t).} \\ \texttt{does(player1, move\_p1\_t2\_1, t+2) :- does(player2, move\_p2\_t1\_1, t+1).} \\ \texttt{does(player1, move\_p1\_t2\_2, t+2) :- not does(player2, move\_p2\_t1\_1, t+1).} \end{array} \right\}$$

In the set above, we will call "*exception rule*" the second rule, and "*general rule*" the third.

Remark 1. The hypothesis space could also be extended so that it accepts two or more exception rules.

$$H = \left\{ \begin{array}{l} \texttt{does(player1, move\_p1\_t0\_1, t).} \\ \texttt{does(player1, move\_p1\_t2\_1, t+2) :- does(player2, move\_p2\_t1\_1, t+1).} \\ \texttt{does(player1, move\_p1\_t2\_2, t+2) :- does(player2, move\_p2\_t1\_2, t+1).} \\ \texttt{does(player1, move\_p1\_t2\_3, t+2) :- not does(player2, move\_p2\_t1\_1, t+1),} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{not does(player2, move\_p2\_t1\_2, t+1).} \end{array} \right\}$$

Moreover, the atoms in the exception rule must appear in the general rule (see the red and blue atoms in the hypotheses above).

With ILASP 3.1.0 it is possible add "bias constraints" to specify what kind of rule we want to learn. However it is not possible to add constraints on the whole set of rules induced. As a consequence, we cannot say that for every solution H in $ILP_{LAS}(B, S_M, E^+, E^-)$, if there are $move\_p1\_t2\_1$ and $move\_p2\_t1\_1$ such that: "does(player1,move_p1_t2_1,t+2) :- does(player2,move_p2_t1_1,t+1)." $\in H$, then there is $move\_p1\_t2\_2$ such that:
"does(player1,move_p1_t2_2,t+2):- not does(player2,move_p2_t1_1,t+1)." $\in H$

The solution we used consists in using the ASPAL encoding in ILASP. With this encoding, we can add more flexible constraints for the hypothesis space.

Good idea in the above to explain why you want to say the global constraint. In other words, what does the constraint impose? Perhaps abstract from the specific case of moves, or at least the specific arguments.

3.1.2 Example

Current State of the Game Consider a tic-tac-toe game after the 4 following moves (represented in 3.1):

```
does(player1, fill(coord(2,2)),1).
does(player2, fill(coord(1,2)),2).
does(player1, fill(coord(1,1)),3).
does(player2, fill(coord(3,3)),4).
```

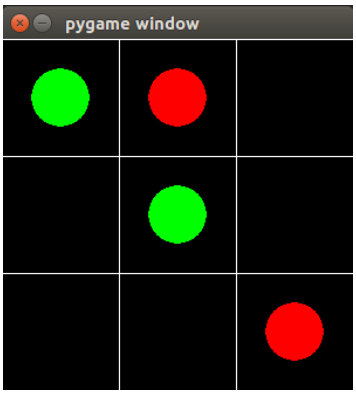

Figure 3.1: State of a tic-tac-toe game after 4 moves. The green player plays first (he is `player1`).

Here, we are the green player and we want to find a move such that we are sure to win in three moves.

Hypothesis space We use the following mode declaration:

$$M = \left\{ \begin{array}{ll} m1: & \texttt{modeh(does(player1,\#move,5)).} \\ m2: & \texttt{modeh(does(player1,\#move,7)).} \\ m3: & \texttt{modeb(does(player2,\#move,6)).} \\ m4: & \texttt{modeb(not does(player2,\#move,6)).} \end{array} \right\}$$

Moreover, we take the following hypothesis space: But lower down you say you want to learn the predicate rule?

$$R_M = \left\{ \begin{array}{l} \texttt{does(player1, X, 5).} \\ \texttt{does(player1, Z, 7) :- does(player2, Y, 6).} \\ \texttt{does(player1, X, 7) :- not does(player2, Y, 6).} \end{array} \right\}$$

So the top theory that we use for this example is: Not sure if these are background clauses?

$$T = \left\{ \begin{array}{l} \texttt{does(player1, X, 5) :- legal(player1,X,5), rule((m1),(X)).} \\ \texttt{does(player1, Z, 7) :- legal(player1, Z, 7), legal(player2, Y, 6),} \\ \qquad\qquad\qquad\quad \texttt{does(player2, Y, 6), rule((m2,m3,2),(Z,Y)).} \\ \texttt{does(player1, X, 7) :- legal(player1, X, 7), legal(player2, Y, 6),} \\ \qquad\qquad\qquad\quad \texttt{not does(player2, Y, 6), rule((m2,m4,2),(X,Y)).} \end{array} \right\}$$

where we added the predicate `legal/3` to make the variables safe.

So now the hypothesis we want to learn with ILASP has the following form:

$$H = \left\{ \begin{array}{l} \texttt{rule((m1),(\#move\_p1\_t0\_1)).} \\ \texttt{rule((m2,m3,2),(\#move\_p1\_t2\_1, \#move\_p2\_t1\_1)).} \\ \texttt{rule((m2,m4,2),(\#move\_p1\_t2\_2, \#move\_p2\_t1\_1)).} \end{array} \right\}$$

We want H to contain only one rule of each type: only one rule of the form `rule((m1),(#move_p1_t0_1))` and so on... So we add some rules to make them unique:

```
rule1 :- rule((m1),(fill(coord(X,Y)))).
:- not rule1.
:- rule((m1),(fill(coord(X1,Y1)))), rule((m1),(fill(coord(X2,Y2)))), g(X1,
Y1)< g(X2, Y2).
```

The two first rules make `rule((m1),(#move_p1_t0_1))` appear at least once, and the last rule makes it appear at most once. We add similar rules for `rule((m2,m3,2), (#move_p1_t2_1, #move_p2_t1_1))` and `rule((m2,m4,2), (#move_p1_t2_2, #move_p2_t1_1))`.

Also, we added some constraints to reduce the computation time. For instance, if `rule((m1),(fill(coord(X,Y))))` is true, then `legal(player1,fill(coord(X,Y)),5)` must be true. So we add the following constraint:

```
:- rule((m1),(fill(coord(X,Y)))), not legal(player1,fill(coord(X,Y)),5).
```

And if `rule((m2,m4,2),(fill(coord(X,Y)),fill(coord(Z,T))))` and `not does(player2, fill(coord(Z,T)), 6)` are true, then `legal(player1, fill(coord(X,Y)),7)` must

be true unless `player2` plays in `coord(X,Y)` at time 6. Thus, we add the following constraint:
`:- rule((m2,m4,2),(fill(coord(X,Y)),fill(coord(Z,T)))), not does(player2, fill(coord(Z,T)), 6), not legal(player1, fill(coord(X,Y)),7).`
and we add this context-dependent example to the set of positive examples :
$<<$ {`wins(player2,8)`}, $\emptyset >$,
{`:- rule((m2,m4,2),(fill(coord(X,Y)),fill(coord(Z,T)))), not does(player2, fill(coord(X,Y)), 6).`} $>$
With this example, we are sure that there will be at least one answer set where `player2` plays in `coord(X,Y)`.

I think it would be good to try to abstract the above first, to make clear the structures you are using to achieve the learning. Could you give an example where the actual moves were just a constant?

Declaring the examples First of all, we want the first player to win in two moves all the time. So there is no answer set that does not contain `wins(player1,8)`. Thus, we take : $E^- = \{< \emptyset, \{\texttt{wins(player1,8)}\} >\}$.

For the positive examples, we only need to take the one given in the previous paragraph. The `wins(player2,8)` in it is optional but it reduces the computation time from 11 seconds to 1 second.

I think someone seeing this for the first time might find it hard to follow. You can then instantiate it with the actual example.

Evaluation

3.1.3 Finding a move in the middle of a game

We are only able to look two moves ahead, so we cannot predict how to win a game if we are not very close to the end. It would be a good idea to define what a good state is (or to make ILASP learn it). And after that, it could be possible to choose a state at time t that maximizes the chances of reaching a good state at time $t+2$.

For instance, if we consider the game represented in figure 3.2, where `player1` plays at times t and $t+2$ and `player2` plays at time $t+1$.
We want to maximize the probability of reaching a good state at $t+2$. So we choose $a(x)$ such that $\#\{b(y)|legal(b(y)) \wedge (b(y) \implies \exists\, c(z)\ ;\ legal(c(z)) \wedge good\_state(state\_c(z)))\}$ is maximal, which is $a(1)$ in this example.

3.2 Second Method: recognize preferred patterns

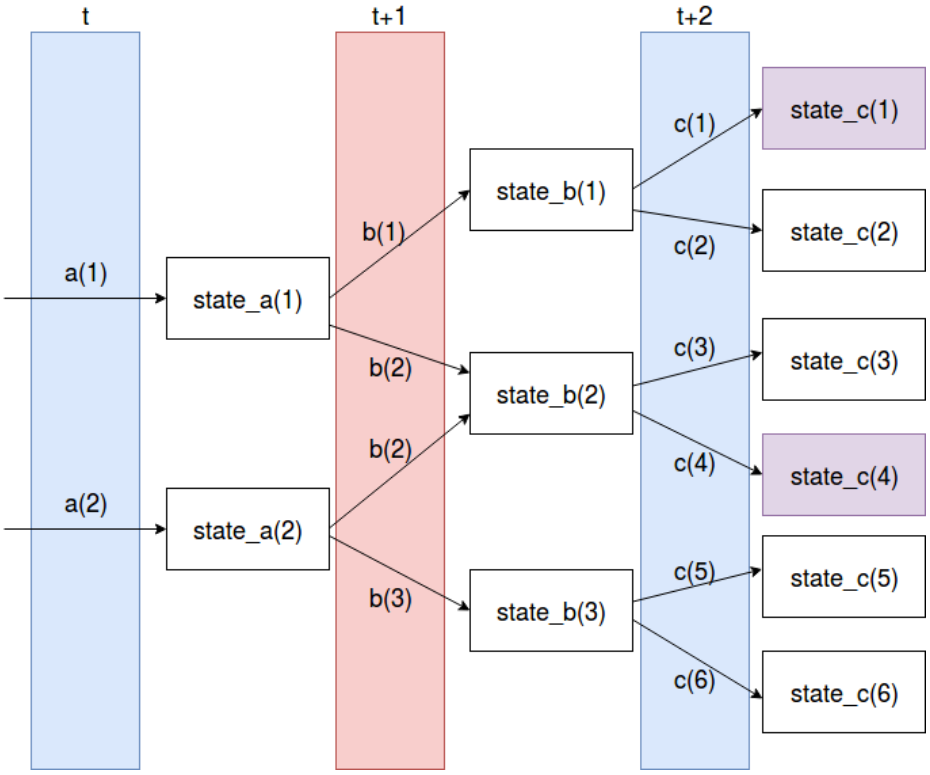

Figure 3.2: Desription of a graph game with its states and the actions to perform to go from one state to another. The blue player (that plays at times t and $t+2$) is `player1`. The states in purple are the good states.