CHAPTER 18

Games with Historical Constraints

18.1 Introduction

One of the distinctive features of GDL is its *Markov* character - the truth values of all state-dependent conditions are defined entirely in terms of the truth values of conditions on the current state and/or the the immediately preceding state. There is no explicit dependence on other states of the world.

In some games and in many real-word applications, it is more convenient to define state-dependent relations (such as legality, reward, and termination) in terms of multiple preceding states. For example, in Chess, we need to express the fact that a player may not castle if it has moved either its king or rook on any preceding step.

Now, for finite games, it is always possible to transform such conditions into Markov conditions by adding information to the state of the game. Unfortunately, this is sometimes inconvenient; and in many real-world applications, this is not possible, since the state description is often controlled by others. In such cases, we need a language to allow us to express non-Markov conditions directly, without such state modifications.

In this chapter, we examine a language called System Definition Language (or SDL), which supports this level of expressiveness. In the next section, we give the details of the language. We then show how to use it in the context of a couple of examples.

18.2 System Definition Language

System Description Language (or SDL) is a non-Markov variant of GDL. Like GDL, descriptions take the form of open logic programs. The only difference is in the game-independent vocabulary, and even this is very similar to that of GDL.

First of all, SDL includes all of the structural relations in GDL without change.

`role(`r`)` means that r is a role in the game.
`base(`p`)` means that p is a base proposition in the game.
`percept(`r`,`p`)` means that p is a percept for role r.
`input(`r`,`a`)` means that a is an action for role r.

To these basic structural relations, we add a couple of relations for talking about steps.

`step(`s`)` means that s is a step.
`successor(`s_1`,`s_2`)` means that step s_1 comes immediately before step s_2.

The relations `true`, `sees`, `does`, `legal`, `goal`, and `terminal` are all the same as in GDL *except that* each is augmented with a step argument.

`true(`p`,`s`)` means that the proposition p is true on step s.
`sees(`r`,`p`,`s`)` means that role r sees percept p on step s.
`does(`r`,`a`,`s`)` means that role r performs action a on step s.
`legal(`r`,`a`,`s`)` means it is legal for role r to play action a on step s.
`goal(`r`,`n`,`s`)` means that player has utility n for player r on step s.
`terminal(`s`)` means that the state on step s is terminal.

And that's it; that's the entire language. We no longer need `init` and `next`. Why? The truth of propositions in the initial state can be stated using `true` with the first step as the step argument; and update rules, formerly written using `next`, can be stated using `true` and `successor`. Just how this works should become clear from the examples given below.

18.3 Example - Tic Tac Toe

As an illustration of SDL, let's see how we can use the language to describe the game of Tic Tac Toe. As we have seen, we can describe the game adequately in GDL. The point of rewriting the description in SDL is to underscore the similarities and differences.

In SDL, as in GDL, we use the ternary function constant `cell` together with a row *m* and a column *n* and a mark *w* to designate the proposition that the cell in row *m* and column *n* contains *w* where *w* is either an `x` or an `o` or a `b` (for blank); and we use the unary function constant `control` to state that it is that role's turn to mark a cell. The binary function `mark` together with a row *m* and a column *n* designates the action of placing a mark in row *m* and column *n*, and the object constant `noop` refers to the act of doing nothing.

The first step in writing an SDL description is the same as that in GDL - we enumerate the structural components of the game - roles, base propositions, percepts (there are none in this case), and actions. In SDL, these descriptions are exactly the same as in GDL.

```
role(white)
role(black)


base(cell(X,Y,W)) :-
  index(X) &
  index(Y) &
  filler(W)

base(control(W)) :-
  role(W)


input(W,mark(X,Y)) :-
  role(W) &
  index(X) &
  index(Y)

input(W,noop) :-
  role(W)


index(1)
index(2)
index(3)

filler(x)
filler(o)
filler(b)
```

To these basic components, we add a set of steps and a suitable successor function. In this case, we use the natural numbers 1, ..., 10 with their usual successor function.

```
step(1)
step(2)
step(3)
step(4)
step(5)
step(6)
step(7)
step(8)
step(9)
```

```
step(10)

successor(1,2)
successor(2,3)
successor(3,4)
successor(4,5)
successor(5,6)
successor(6,7)
successor(7,8)
successor(8,9)
successor(9,10)
```

The initial state of the game is written using the `true` relation and the first step (in this case the natural number 1).

```
true(cell(1,1,b),1)
true(cell(1,2,b),1)
true(cell(1,3,b),1)
true(cell(2,1,b),1)
true(cell(2,2,b),1)
true(cell(2,3,b),1)
true(cell(3,1,b),1)
true(cell(3,2,b),1)
true(cell(3,3,b),1)
true(control(white),1)
```

The update rules are basically the same as in GDL. There are just two differences. (1) We add a step argument to `does` and `true`. (2) We replace `next` with `true`. (3) We use `successor` to relate successive steps.

```
true(cell(I,J,x),N) :-
  does(white,mark(I,J),M) &
  successor(M,N)

true(cell(I,J,o),N) :-
  does(black,mark(I,J),M) &
  successor(M,N)

true(cell(K,L,b),N) :-
  does(W,mark(I,J),M) &
  true(cell(K,L,b),M) &
  distinct(I,K) &
  successor(M,N)

true(cell(K,L,b),N) :-
  does(W,mark(I,J),M) &
  true(cell(K,L,b),M) &
  distinct(J,L) &
  successor(M,N)

true(cell(I,J,Z),N) :-
  does(W,mark(I,J),M) &
  true(cell(I,J,Z),M) &
  distinct(Z,b) &
  successor(M,N)

true(control(white),N) :-
  true(control(black),M) &
  successor(M,N)

true(control(black),N) :-
  true(control(white),M) &
  successor(M,N)
```

Our view relations are the same except that, once again, we add step arguments. In this case, we do not need the `successor` relation since the same step is involved in all cases.

```
row(M,W,S) :-
  true(cell(M,1,W),S) &
  true(cell(M,2,W),S) &
  true(cell(M,3,W),S)

column(N,W) :-
  true(cell(1,N,W)) &
  true(cell(2,N,W)) &
  true(cell(3,N,W))

diagonal(W) :-
  true(cell(1,1,W)) &
  true(cell(2,2,W)) &
  true(cell(3,3,W))

diagonal(W) :-
  true(cell(1,3,W)) &
  true(cell(2,2,W)) &
  true(cell(3,1,W))

line(W,N) :- row(J,W,N)
line(W,N) :- column(K,W,N)
line(W,N) :- diagonal(W,N)

open(N) :- true(cell(J,K,b),N)
```

Finally, we define the notions of legality, goal, and termination using our augmented versions of these relations.

```
legal(W,mark(X,Y),N) :-
    true(cell(X,Y,b),N) &
    true(control(W),N)

legal(white,noop,N) :-
    true(control(black),N)

legal(black,noop,N) :-
    true(control(white),N)

goal(white,100,N) :- line(x,N) & ~line(o,N)
goal(white,50,N) :- ~line(x,N) & ~line(o,N)
goal(white,0,N) :- ~line(x,N) & line(o,N)

goal(black,100,N) :- ~line(x,N) & line(o,N)
goal(black,50,N) :- ~line(x,N) & ~line(o,N)
goal(black,0,N) :- line(x,N) & ~line(o,N)

terminal(N) :- line(W,N)
terminal(N) :- step(N) & ~open(N)
```

And that's it. Everything is pretty much the same as before. We have just made explicit the Markov character of the game description.

18.4 Example - Chess

In order to see how we can use SDL to encode non-Markov constraints, let's look at the game of Chess and, in particular, the legality conditions for castling and *en passant* captures. While it is possible to describe these conditions in GDL, that description requires that we include some base propositions specifically for this purpose. In SDL, we can describe the conditions directly, without these additional base propositions.

As in our description of Tic Tac Toe, we begin with a description of the structural elements of the game - roles, base propositions, percepts (again none), actions, and steps. Here, we have limited the game to 100 steps.

```
role(white)
role(black)

base(cell(I,J,W)) :-
  file(I) &
  rank(J) &
  filler(W)

base(control(W)) :-
  role(W)


input(W,move(I,J,K,L)) :-
  role(W) &
  file(I) &
  rank(J) &
  file(K) &
  rank(L)

input(W,castleshort) :-
  role(W)

input(W,castlelong) :-
  role(W)

input(W,noop) :-
  role(W)

file(a)
file(b)
file(c)
file(d)
file(e)
file(f)
file(g)
file(h)

rank(1)
rank(2)
rank(3)
rank(4)
rank(5)
rank(6)
rank(7)
rank(8)

filler(wk)
filler(wq)
filler(wb)
filler(wn)
filler(wr)
filler(wp)
filler(bk)
filler(bq)
filler(bb)
filler(bn)
filler(br)
filler(bp)
filler(b)

step(1)
step(2)
  ...
step(99)
step(100)

successor(1,2)
successor(2,3)
  ...
```

```
successor(98,99)
successor(99,100)
```

There two types of castling in Chess - castling on the king side (`castleshort`) and castling on the queen side (`castlelong`). In either case, the corresponding rook is moved to the cell adjacent to the king, and the king is placed on the other side of the rook.

Castling is a very useful move in many situations. However, it can only be done if certain conditions are met. First, the king and the rook must be in their original positions and must not have been previously moved. Second, the spaces between the king and the rook must be clear. Third, the spaces from the initial position of the king to the final position must not be under attack.

Let's formalize the first of these conditions in the case of white castling on the king side. (The other conditions pose no special difficulties; and, for the sake of simplicity, we ignore them.)

The corresponding definition of legality for `white` to `castleshort` is shown below. The move is legal provided that the king and the rook are on their initial squares and the squares in between are empty and neither the king nor the rook have been moved.

```
legal(white,castleshort,N) :-
  true(cell(e,1,wk),N) &
  true(cell(f,1,b),N) &
  true(cell(g,1,b),N) &
  true(cell(h,1,wr),N) &
  ~moved(wk,N) &
  ~moved(wr,N)
```

The white king has been moved on step `N` if there was any move from cell `e1` on that step or any preceding step. The king-side white rook has been moved on step `N` if there was any move from cell `h1` on that step or any preceding step. Finally, a piece has been moved on a step if it was moved on any preceding step.

```
moved(wk,N) :-
  does(W,move(e,1,K,L),N)

moved(wr,N) :-
  does(W,move(h,1,K,L),N)

moved(P,N) :-
  moved(P,M) &
  successor(M,N)
```

This definition of legality guarantees that, if either the white king or the king-side white rook have been moved, then castling on the king side is not legal. Significantly, the condition is stated using just the state of the board and the history of moves, without any extraneous propositions in the state description.

Now, on to *en passant* captures. If a pawn has not been moved, a player has the option of advancing the pawn either one or two steps forward. If it chooses to advance its pawn two steps, it may move through a cell which is threatened by an opposing pawn. In this case, the opponent has the option of taking the pawn even though it is no longer on a threatened square.

The conditions for *en passant* captures are strict. The player must have moved the pawn for the first time, and it must have made that move on the preceding step. We can formalize these conditions in SDL as shown below. Here, for simplicity, we have shown the conditions for the black player to make the *en passant* capture, and we have shown only the case of a pawn move on the `e` file and a capture from the f file. (Expanding this for the general case is messy but straightforward.)

```
legal(black,move(f,4,e,3),N) :-
  does(white,move(e,2,e,4),M) &
  successor(M,N)
```

Although, in this example, the rule concerns only two states, it still requires SDL because it defines legality in one state on the basis of a move in the preceding state rather than on conditions in the current state, as in GDL.

This definition guarantees the desired legality conditions; and, as with the preceding example, the condition is stated using just the state of the board and the history of moves, without any extraneous propositions in the state description.