REASONING IN DYNAMIC DOMAINS

So far we considered only knowledge bases describing static unchanging worlds. In this section we discuss how to represent the change. We will start with a simple example and continue to build a methodology of representing knowledge in dynamic domains.

The Blocks World

Consider a collection of blocks, $b_1, \ldots, b_n$, forming towers on table t as in Assignment 2. In addition let us imagine a robot's arm capable of moving block B to location L. Of course the arm can only perform this action if both, B and L are clear.

We would like to be able to describe the transformation of the domain caused by the arm's activity, i.e. given an initial positions σ_0 of blocks and a sequence α of actions we should be able to predict the position of blocks after the execution of α.

Objects of the Domain

1. Discrete Time (SMODELS syntax)

$const\ n = 8.$ $step(0..n)$**.**

2. As before we will have blocks and locations which do not depend on time:

(a) Blocks: $block(a)$**.** $block(b)$**.** $block(c)$**. etc.**

(b) Locations: (blocks and the table)

$location(L)$ **:-** $block(L)$**.**

$location(t)$**.**

3. Relation on **however does depend on time and hence acquire an extra parameter,** T**. Relations whose truth value depends on time are often called 'fluents'.**

$on(B, L, T)$ **- at time** T **block** B **is located on** L**.**

4. Finally we have actions: $put(B, L, T)$ **- at time** T**,** B **is put on** L**.**

The Initial Configuration

1. initial configuration of blocks, e.g.

```
 2       7

 3   4   6

 0   1   5

----------
```

can be described by

(a) collection of atoms

$on(b0, t, 0).\ on(b3, b0, 0).\ on(b2, b3, 0).\ on(b1, t, 0).$

$on(b4, b1, 0).\ on(b5, t, 0).\ on(b6, b5, 0).\ on(b7, b6, 0).$

and closed world assumption

$-on(B, L, 0)$ **:-** $not\ on(B, L, 0)$

B **stands for blocks,** L **for locations and** T **for time.**

Effects of Actions

Let us assume that every action takes one unit of time. Then the axiom

$$1. \quad on(B, L, T+1) \textbf{ :- } put(B, L, T).$$

describes the effect of put**. We assume here that the robot never drops the block and is otherwise perfect 'executor' of action.**

Rule (1) can be viewed as a special case of a dynamic causal law - statement of the form

$$a \textbf{ causes } f \textbf{ if } p$$

which says that action a **executed in a state satisfying conditions** p **causes fluent** f **to become true in the resulting state.**

Effects of Actions

The new location of B can be viewed as a 'direct' effect of action put. There are also 'indirect' effects caused by relationships between fluents of the domain, e.g. a block can only occupy a single location, and no block can support more than two other blocks. This can be expressed by axioms

2. $-on(B, L_1, T)$ **:-** $on(B, L_2, T),\ L_1 \neq L_2.$

3. $-on(B_2, B, T)$ **:-** $on(B_1, B, T),\ B_1 \neq B_2.$

Rules (2,3) can be viewed as a special case of a static causal law (also referred to as state constraints) - statement of the form

$$f \textbf{ if } p$$

which says that every state satisfying p must satisfy f.

Effects of Actions

The above causal laws describe the changes caused by execution of an action. We also need to describe what is not changed. To do that we introduce a predicate $moved(B, T)$ - 'block B is moved at time T':

$$4a.\ \ moved(B, T) \textbf{ :- } put(B, L, T).$$

The rules

$$4b.\ on(B, L, T + 1) \textbf{ :- } on(B, L, T), not\ moved(B, T).$$

$$4c.\ -on(B, L, T+1) \textbf{ :- } -on(B, L, T), not\ moved(B, T).$$

say that positions of all other blocks are unchanged. Axioms (1–4) give a complete description of a state σ_1 which results from executing action $put(B, L, T)$ in state σ_0. Later we study how to define 'successor' states in more complex domains.

Effects of Actions

The next two axioms are constraints on executability of actions.

```
5a.     :-  put(B1,B,T),
            on(B2,B,T).
5b.     :-  put(B,L,T),
            on(B1,B,T).
```

Now we have a simple theory of the blocks world. To extract the state of the world after block 2 is moved on the table and 7 is put on 2 we need to compute the answer set of the above program, BW, augmented by

$$put(2,t,0). \qquad put(7,2,1).$$

Atoms $on(B,L,2)$ will give complete description of the new state.

Using the Theory: Planning

Given: Planning Problem

1. initial configuration of blocks, e.g.

```
 2       7

 3   4   6

 0   1   5

----------
```

2. final configuration of blocks, e.g.

```
 7       5

 3   2   0

 4   6   1

----------
```

Find: a plan (sequence of actions) to move from initial to final configuration.

Describing the Goal

There are many different ways to represent a final configuration. We use a simple rule

```
goal(T) :-
   on(4,t,T),  on(6,t,T),   on(1,t,T),
   on(3,4,T), on(7, 3,T), on(2,6,T),
   on(0,1,T), on(5,0,T).
```

defining the goal state T. To achieve the goal we need to find such T, i.e. to satisfy the two rules below:

```
success :- step(T),goal(T).
:- not success.
```

Generating the Candidate Plans

Recall that to use answer set programming methodology we had to restrict ourselve to a finite time interval $[0..n]$, This means that we can only look for plans of no more then n consequitive steps. Candidate plans will be generated by choice rule:

```
1{put(B,L,T) : block(B) : loc(L)}1 :- step(T),
                                      not goal(T),
                                      T < n.
```

For any step from 0 to n-1 which does not satisfy the goal the rule will select a candidate action. Answer sets of the program will correspond to plans, i.e. sequences of actions satisfy *success*.

The Program's Output

Statements $hide.$ **and** $show\ put(B, L, T).$

extract the plans from the corresponding answer sets. Here is the output of the call:

$lparse\ --true-negation\ file \mid smodels \mid mkatoms$

$put(7, t, 0)\ put(4, t, 1)\ put(6, t, 2)\ put(2, 6, 3)$

$put(3, 4, 4)\ put(7, 3, 5)\ put(0, 1, 6)\ put(5, 0, 7)$

Run the program with

(a) smodels 0

(b) n = 9

and explain the results.

Comments

• The above program, $P1$, is a typical example of Answer Set Planning. The program consists of the theory of blocks world, description of the goal, and 'the planning module' - in our case a simple choice rule. Note that the blocks theory is independent from planning - it can be used for multiple purposes.

• Answer set planning does not require any specialized planning algorithm. The 'planning' query is answered by the reasoning mechanism used for other types of queries.

• The planning program can be easily generalized and improved.

Modifying the Planner – Concurrent Actions.

So far we dealt with one arm domain. Suppose now that we have two robot arms capable of avoiding collisions in the air. Let us show that the planner can be easily modified to allow for this situation and hence for parallel actions.

To adopt $P1$ to this case we need to:

1. add more constraints needed to allow parallel actions. In particular we prohibit putting things on moving blocks.

```
:- block(B1),block(B2),
   loc(L), step(T),
   put(B1,L,T),
   put(B2,B1,T).
```

2. replace our old 'generator' rule by the new one:

```
1{put(B,L,T) : block(B) : loc(L)}2 :- step(T),
                                      not goal(T),
                                      T < n.
```

By decreasing the number of steps in the program we will be able to find plans of better and better quality, with a significant improvement in efficiency of search. With n=5 SMODELS produced a plan;

$put(2,t,0); put(4,t,0);$

$put(3,4,1); put(7,t,1);$

$put(0,1,2); put(6,t,2);$

$put(2,6,3); put(5,0,3);$

$put(7,3,4).$

There is no plan for n=4.

Adding Heuristic Information

The previous solutions didn't contain any information distinguishing between 'good' and 'bad' moves of blocks. Information of this sort is often called heuristics or control information. In many cases it helps to dramatically reduce the search space of the problem. Let us supply our sequential search problem in the blocks world with DO NOT DESTROY A GOOD TOWER heuristics. To encode this heuristics in A-Prolog we expand the problem description by a new representation of the goal

```
want(4,t).  want(6,t).  want(1,t).
want(3,4). want(7,3). want(2,6).
want(0,1). want(5,0).
```

Do not destroy good towers

The relation $out\_of\_place(B,T)$ holds if after T steps block B is not yet put in its proper position. As always, B's, L's and T's in the rules below stand for blocks, locations and steps.

```
out_of_place(B,T) :- want(B,L),
                     not on(B,L,T).
out_of_place(B,T) :- on(B,B1,T),
                     out_of_place(B1,T).
out_of_place(B2,T) :- want(B1,B),
                      on(B2,B,T),
                      neq(B2,B1).
:- not out_of_place(B,T),
   put(B,L,T).
```

Test the heuristics. Invent and encode other heuristics.