CS348

Summer 2001

MP 3: Planning in Prolog

Due: Thursday, July 19

Joseph S Kim

DESCRIPTION

This machine problem consists of writing small planning programs in Prolog. There are two tasks to be completed in block's world, Sussman Anomaly and Block Painting, by given basic STRIPS operators such as move and some utilities for stack, queue, and set operations ('adts.pl', 'planner.pl').

IMPLEMENTATION

1. Sussman Anomaly

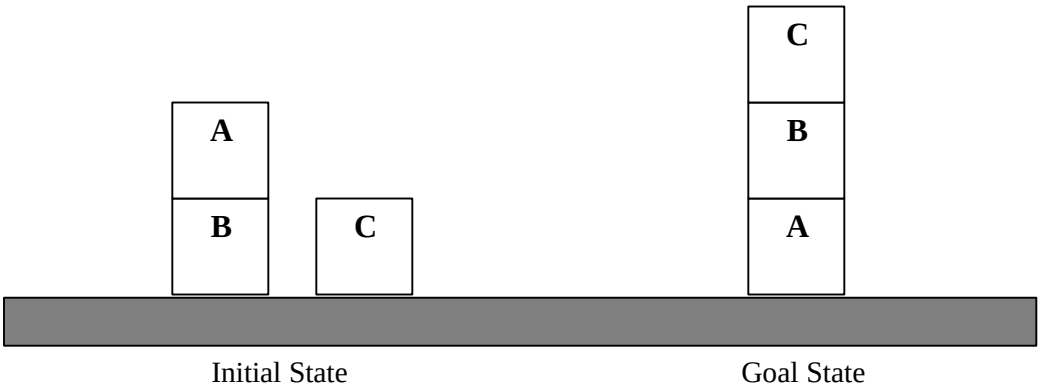

Figure 1: Sussman Anomaly

Initial State: handempty & clear(A) & clear(C) & on(A, B) & ontable(B) & ontable(C)
Goal State: on(C, B) & on(B, A)

If STRIPS is presented with a goal consisting of a conjunction of predicates, it solves each goal conjunct or predicate one at a time in order. A plan generated by this method contains a sequence of actions for attaining the first goal conjunct, followed by a sequence of actions for attaining the second, and so on.

However, some problems, such as the Sussman Anomaly contain interactions. For example, in order to achieve the second goal conjunct on(B, A) we have to undo the goal on(C, B). In

other words, the actions that are used to solve one sub-problem or goal conjunct, may interfere with the solution to a previous sub-problem or goal conjunct. STRIPS is therefore unable to produce optimal solutions to problems in which interactions occur.

- if we tackle On(C,B) goal first, we will likely undo it as we try to achieve On(B,A)
- similarly, if we tackle On(B,A) first, we will undo it with On(C,B)

Both progression and regression planners are capable of handling the Sussman Anomaly, but the amount of effort required seems inordinate. This is because the planners must work in a linear fashion, adding steps that build on their immediate successors or predecessors.

- STRIPS is known as a **linear planner (total-order)** because it represents plans as a totally ordered sequence of actions
- STRIPS also assumes that subplans cannot be interleaved. This is known as the **linearity assumption**
- Subplans are merged together by concatenation
- This prevents STRIPS from solving certain problems, e.g., **register exchange problem**

- Predicates for init_state() and goal()

```
% init_state()
init_state(q1, [clear(a),clear(c),ontable(b),ontable(c),on(a,b),
handempty]).
% goal()
goal(q1, [on(c,b), on(b,a), clear(c), ontable(a)]).
```

2. Blocking Painting

```
move(paint(X,Y), [ontable(X), clear(X), holding(brush(Y))],
[add(color(X,Y))]).

init_state(q3p1, [clear(a),ontable(a),clear(brush(red)),ontable(brush(red)),
handempty]).
goal(q3p1, [color(a,red)]).

init_state(q3p2,
[clear(a),ontable(a),clear(b),ontable(b),clear(brush(red)),ontable(brush(red
)), handempty]).
goal(q3p2, [on(a,b),color(a,red)]).

init_state(q4,
[clear(a),on(a,b),ontable(b),clear(brush(blue)),ontable(brush(blue)),clear(b
rush(red)),ontable(brush(red)), handempty]).
goal(q4, [on(a,b),color(a, blue),color(b,red)]).
```

During the planning process several alternative nodes are generated containing a representation of the current state; a goal stack; a set of actions. At each point it is necessary to find heuristics that enable the best node to be chosen for subsequent refinement. Domain independent planning algorithms can use heuristics based on minimizing the number of outstanding goals on the goal stack as well as the number of actions contained in each node (i.e. a good node to choose for expansion is one with few outstanding goals to be achieved and with few actions).

Alternatively, domain specific information could be used. For example, if we knew how long it would take to execute actions, a good node for expansion would be one that contained actions that took a small amount of time to execute. In the blocks world domain, we also know that it is possible to end up with a plan containing an infinite cycle of stack and un-stack actions - domain specific information could consist of detecting and not choosing nodes, which contain such cycles. If we relocate move() predicate to the bottom of goal() predicate, then running time is actually decreased by 20ms.

These are results from each predicates.

```
| ?- test(q1).
moves are
pickup(a)
putdown(a)
pickup(b)
stack(b,a)
pickup(c)
stack(c,b)

(10 ms) yes


| ?- test(q3p1).
moves are
pickup(brush(red))
paint(a,red)

yes


| ?- test(q3p2).
moves are
pickup(brush(red))
paint(a,red)
putdown(brush(red))
pickup(a)
stack(a,b)

(10 ms) yes


| ?- test(q4).
moves are
pickup(a)
stack(a,brush(blue))
pickup(b)
stack(b,brush(red))
pickup(a)
putdown(a)
pickup(b)
stack(b,a)
pickup(brush(red))
stack(brush(red),brush(blue))
pickup(b)
putdown(b)
pickup(brush(red))
stack(brush(red),b)
pickup(brush(blue))
stack(brush(blue),a)
pickup(brush(red))
putdown(brush(red))
pickup(brush(blue))
stack(brush(blue),brush(red))
pickup(a)
stack(a,b)
pickup(brush(blue))
paint(brush(red),blue)
stack(brush(blue),brush(red))
pickup(a)
putdown(a)
```

```
pickup(brush(blue))
stack(brush(blue),a)
pickup(brush(red))
stack(brush(red),b)
pickup(brush(blue))
putdown(brush(blue))
pickup(brush(red))
stack(brush(red),brush(blue))
pickup(b)
stack(b,a)
pickup(brush(red))
putdown(brush(red))
pickup(b)
stack(b,brush(red))
pickup(a)
stack(a,brush(blue))
pickup(b)
putdown(b)
pickup(a)
stack(a,b)
pickup(brush(red))
paint(brush(blue),red)
stack(brush(red),brush(blue))
pickup(a)
putdown(a)
pickup(brush(red))
stack(brush(red),a)
pickup(brush(blue))
stack(brush(blue),b)
pickup(brush(red))
putdown(brush(red))
pickup(brush(blue))
stack(brush(blue),brush(red))
pickup(b)
stack(b,a)
pickup(brush(blue))
putdown(brush(blue))
pickup(b)
stack(b,brush(blue))
pickup(a)
stack(a,brush(red))
pickup(b)
putdown(b)
pickup(a)
stack(a,brush(blue))
pickup(brush(red))
stack(brush(red),b)
pickup(a)
putdown(a)
pickup(brush(blue))
stack(brush(blue),a)
pickup(brush(red))
putdown(brush(red))
pickup(b)
stack(b,brush(red))
pickup(brush(blue))
paint(a,blue)
stack(brush(blue),a)
pickup(b)
putdown(b)
pickup(brush(blue))
stack(brush(blue),b)
pickup(a)
```

```
stack(a,brush(red))
pickup(brush(blue))
putdown(brush(blue))
pickup(a)
stack(a,brush(blue))
pickup(brush(red))
stack(brush(red),b)
pickup(a)
putdown(a)
pickup(brush(red))
stack(brush(red),a)
pickup(b)
stack(b,brush(blue))
pickup(brush(red))
paint(a,red)
stack(brush(red),a)
pickup(b)
putdown(b)
pickup(brush(red))
stack(brush(red),b)
pickup(a)
stack(a,brush(blue))
pickup(brush(red))
putdown(brush(red))
pickup(a)
stack(a,brush(red))
pickup(brush(blue))
stack(brush(blue),b)
pickup(a)
putdown(a)
pickup(brush(blue))
stack(brush(blue),a)
pickup(brush(red))
paint(b,red)
stack(brush(red),b)
pickup(brush(blue))
putdown(brush(blue))
pickup(brush(red))
stack(brush(red),brush(blue))
pickup(b)
stack(b,brush(red))
pickup(a)
stack(a,b)

(710 ms) yes
```