A Planner Called R

Fangzhen Lin (flin@cs.ust.hk)
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

System R was competed at the AIPS-2000 competition, and it was the only planner that was competed in both the automatic and hand-tailered tracks. This article is a short and informal description of this planner.

Introduction

System R is a variant of the original STRIPS by Fikes and Nilsson (1971) (called forward-chaining, recursive STRIPS in (Nilsson 1998)). Briefly speaking, given a conjunction of simple goals, it selects one of them to work on. If there is an action that is executable, and will make this simple goal true once executed, then it does this action and then tries to achieve the other simple goals in the new situation. Otherwise, it computes a new conjunctive goal based on the actions that have this simple goal as one of their effects, and recursively tries to achieve the new goal. This process continues until the original goal is achieved.

The main differences between system R and STRIPS (as given in (Nilsson 1998) on pages 377-378) are as follows:

- STRIPS maintains a global database which on starting up is the initial state description. During the planning process, it updates this global database so that it is always the description about the current situation. This means that once STRIPS finds an action to apply, it commits to it, and no backtracking can undo that. In comparison, our algorithm does not modify the initial state description. Instead, it maintains a global list Γ that represents the sequence of actions that the planner has found so far at the moment. But on backtracking, as for example on reaching a deadend when pursuing a simple goal, some of the actions in Γ may be backtracked. This makes our planner more flexible. On the other hand, this also means that checking if a fluent is true in the current situation is costlier in our algorithm than in STRIPS.
- When attempting to achieve a simple goal, our algorithm keeps track of the sequence of all the simple goals that the planner has subgoaled to and makes sure that no cycles are involved. That is, it does not do something like "to achieve g, try to achieve g first." This seems to be a crucial factor in making our algorithm effective on some of domains at the competition.
- When attempting to achieve a simple goal g that cannot be made true immediately by an executable action, STRIPS selects an action that can make g true, and subgoals it to the preconditions of this action. In general, some of these conditions may contain variables, and these variables are eventually grounded by matching them with the current state description. For our planner, however, the user can decide how to achieve a simple goal using domain specific information. We will describe this in more detail later in the section on controlling the planner. If there are no user provided information, the planner uses the following strategy:

  1. compute first the common ground preconditions of all ground actions that can make g true. If any of them is not yet true in the current situation, then the conjunction of these conditions is chosen to be the new goal. If all of them are true already, then it performs the next step.
  2. selects an action that can make g true; consider first the conjunction of all *ground* preconditions of this action. If this conjunction is not yet true in the current situation, then choose it as the new goal. If it is already true, then let the new goal be the conjunction of all the preconditions of this action with all the variables instantiated by some constants.

  Notice that in all cases, the new goal is always grounded.

The blocks world

We now illustrate our planning algorithm with an example from the blocks world. In the version given at the AIPS-2000 competition, there are four actions, given below in STRIPS style:

```
pickup(x)
  PC: handempty, clear(x), ontable(x)
  D:  handempty, clear(x), ontable(x)
  A:  holding(x)
```

```
putdown(x)
  PC: holding(x)
  D:  holding(x)
  A:  handempty, clear(x), ontable(x)
stack(x,y)
  PC: holding(x), clear(y)
  D:  holding(x), clear(y)
  A:  on(x,y), clear(x), handempty
unstack(x,y)
  PC: handempty, on(x,y), clear(x)
  D:  handempty, on(x,y), clear(x)
  A:  holding(x), clear(y)
```

Our algorithm is sensitive to the order of actions. For now, we assume that the order is as above, which is the same order given at the AIPS-2000 competition. Now consider the following initial state description:

$$on(c, b), on(b, a), ontable(a), clear(c), handempty$$

and the goal $clear(a)$. Figure 1 shows the trace of our planning algorithm on this problem. Begin with the top goal $clear(a)$ in the *Goal* column, it first pushes $clear(a)$ to the stack Π of simple goals at step 2 and preceeds to find a plan for this simple goal. Since there are no common preconditions among all the actions that can possibly make it true, it finds the first action that can make $clear(a)$ true, which is $putdown(a)$ in the *Action* column, and makes the only (ground) precondition of this action, which is $holding(a)$, the new goal and records it in *Goal* column. In step 4, $holding(a)$ is pushed to Π and becomes the current simple goal for the planner to achieve. In step 5, the planner finds out that among all actions that can possibly make $holding(a)$ true, $handempty, clear(a)$ are the common preconditions, so it makes their conjunction the new goal to achieve. But then it finds out that one of the conjuncts, $clear(a)$ is already in Π, so it fails the simple goal $holding(a)$ in step 6, and goes back to step 2 to find alternative actions for achieving $clear(a)$, which it finds to be $stack(a, x)$ for some x. But the ground precondition for this action is $holding(a)$ (as indicated in the *Goal* column), which cannot possibly succeed as the previous steps shows, so it goes back to step 2 to find another alternative action for achieving $clear(a)$, and this time it finds $unstack(x, a)$ for some x in step 9. Since the common precondition of these actions is $handempty$ which is already true in the initial situation, so it begins to instantiate x according to some pre-set order on the blocks, which in our examples is $[a, b, c]$. Step 10 shows that $x = a$ will not work as one of the precondition of $unstack(a, a)$ would require $clear(a)$ which is already in Π. So it tries $x = b$ in step 12 and makes its preconditions, $handempty, on(b, a), clear(b)$, the new goal to achieve. Since the first two simple goals are already true in the current state, which is the initial state at this point, so it puts $clear(b)$ into Π. Now the planner tries to achieve $clear(b)$, and evetually returns the plan $[unstack(c, b), putdown(c), unstack(b, a)]$. Notice that in step 26, after $clear(b)$ is acheived with action $unstack(c, b)$, it remembers that this simple goal is part of the conjunctive goal in step 12, so it restores this conjunctive goal in the next step (step 27) as the current conjunctive goal to achieve and finds out that after this action, $handempty$ is no longer true, so it pushes it to Π in step 28. Step 29 returns $putdown(c)$ as the action for achieving this simple goal as it is the first action that is executable and will make it true once executed.

Problem	Time	Plan length
20-0	9.26	72
25-0	27.46	90
30-0	67.41	104
35-0	146.63	128
40-0	286.96	146
45-0	523.09	174

Table 1: The blocks world. The first number in Problem column is the number of blocks in the problem; times are CPU seconds.

Post-Processing

While the planner returns a reasonable plan in the above example, this may not be the case in general. For instance, given the same initial situation description, and the goal $on(a, b), on(b, c)$, it will return the following plan:

$$[unstack(c, b), putdown(c), unstack(b, a), putdown(b), pickup(a), stack(a, b), unstack(a, b), putdown(a), pickup(b), stack(b, c), pickup(a), stack(a, b)]$$

which is obviously not a very good one. For the AIPS-2000 competition, we implemented the following strategy for post-processing:

> Remove all immediate cycles: If an action is followed immediately by its complement, then remove both of the actions.

Here an action B is a complement of action A if B always restore whatever changes made by A. For instance, in the blocks world, $pickup(x)$ and $putdown(x)$ are complementary to each other, and so are $stack(x, y)$ and $unstack(x, y)$. It is clear that this strategy is always correct and will return a plan that is at least as good as the original one. As it turned out, this is a very effective strategy for the blocks world. For instance, after post-processing, the above plan becomes:

$$[unstack(c, b), putdown(c), unstack(b, a), stack(b, c), pickup(a), stack(a, b)]$$

Table 1 shows the performance of our planner on some typical blocks world problems at the competition.

There are some other strategies one can use. We are currently experimenting with one that will maximize the number of actions that can be executed concurrently and do them as early as possible.

Controlling the planner

The performance of this planner depends on the following factors:

- For each fluent F, the order of the actions that can make F true. For instance, in the blocks world, when attempting the simple goal $clear(x)$, a lot of backtracking can be avoided if we order $unstack(y, x)$ before other actions.
- The ordering of simple goals in a conjunctive goal, in particular, the ordering of preconditions of actions.

While it is easy to provide for some mechanisms for the user to control these orderings explicitly, we do not do so for our planner because we believe this is not the level where the user wants to get her hands on. Rather, the burden should be on the planner to come up with a good ordering by analysizing the domain at hand. This is an area that we are trying to improve on for the next version of the planner.

The main mechanism that we provide for the user to control the planner with domain specific information is through a SOLVE-A-GOAL predicate that would override the default strategy used by the planner for trying to achieve a simple goal. Given a simple goal g, a situation s, SOLVE-A-GOAL$(g, Action, Goal, s)$ means that to achieve g in s, either do $Action$ or (when $Action = nil$) try to achieve $Goal$ first. For instance, for the blocks world, we can write:

- To clear a block, clear the one that is on it:

```
solver-a-goal(clear(X), nil,[clear(Y)],s) :-
  holds([on(Y,X)],s), \+ holds([clear(Y)],s).
```

- To make the hand empty, drop the one that the robot is holding:

```
solver-a-goal(handempty, putdown(X),[],s) :-
  holds([holding(X)],s).
```

- To hold a block, clear it first (if it is already clear, there must be an action that is executable and can make it true, so it would not come to this step):

```
solver-a-goal(holding(X), nil, [clear(X)],s).
```

These rules, together with a pre-processing step that would sort the top level goal according to the order:

$$[ontable(x_n), on(x_{n-1}, x_n), ..., on(x_2, x_1)],$$

are the control strategies used by our planner at the AIPS-2000 competition. Table 2 shows the performance of these control rules on some typical problems from the competition.

Public availability of the system

System R is distributed free of charge and without warranty at the following web site:

http://www.cs.ust.hk/~flin

Problem	Time	Plan length
80-0	1.52	296
100-0	2.49	368
200-0	9.73	732
300-0	22.71	1158
500-0	65.35	1962

Table 2: The blocks world with control information: see Table 1 for legends.

References

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

N. J. Nilsson. *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann Publishers, Inc., San Francisco, CA., 1998.

	Actions	Π (Stack of simple goals)	Goal	Γ (Plans)
1.			clear(a)	
2.		clear(a)		
3.	putdown(a)	clear(a)	holding(a)	
4.		holding(a), clear(a)		
5.	pickup(a) or unstack(a,x)	holding(a), clear(a)	handempty, clear(a)	
6.		fail, go back to 2.		
7.	stack(a,x)	clear(a)	holding(a)	
8.		fail (like 3), go back to step 2.		
9.	unstack(x,a)	clear(a)	handempty	
10.	unstack(a,a)	clear(a)	handempty, on(a,a), clear(a)	
11.		fail, go back to step 9		
12.	unstack(b,a)	clear(a)	handempty, on(b,a), clear(b)	
13.		clear(b), clear(a)		
14.	putdown(b)	clear(b), clear(a)	holding(b)	
15.		holding(b), clear(b), clear(a)		
16.	pickup(b) or unstack(b,x)	holding(b), clear(b), clear(a)	handempty,clear(b)	
17.		fail, go back to 13.		
18.	stack(b,x)	clear(b), clear(a)	holding(b)	
19.		fail (like 16), go back to 13		
20.	unstack(x, b)	clear(b), clear(a)	handempty	
21.	unstack(a, b)	clear(b), clear(a)	handempty, on(a,b), clear(a)	
22.		fail, go back to 20		
23.	unstack(b, b)	clear(b), clear(a)	handempty, on(b,b), clear(b)	
24.		fail, go back to 20		
25.	unstack(c, b)	clear(b), clear(a)	handempty, on(c,b), clear(c)	
26.		clear(a)		unstack(c,b)
27.		clear(a)	handempty, on(b,a), clear(b) (from 12)	unstack(c,b)
28.		handempty, clear(a)		unstack(c,b)
29.		handempty, clear(a)		unstack(c,b), putdown(c)
30.		clear(a)	handempty, on(b,a), clear(b) (from 12)	unstack(c,b), putdown(c)
31.		clear(a)		unstack(c,b), putdown(c), unstack(b,a)

Figure 1: Execution trace for the goal clear(a).