

Materials:           handout of blocks world axioms for proof  
                  handout and demo of BLOCKS.PRO (in demos.prolog)

## I. Introduction

- -----

- A. One important task that arises in many AI systems is the solving of PLANNING PROBLEMS: devising a series of actions to accomplish some goal.
  - 1. An early area of interest, but one where there are still many unresolved problems.
  - 2. This area of work is very important in robotics, but can also be applied to other areas such as puzzle-solving.
- B. Historically, planning work has been closely related to work in automatic theorem proving.
  - 1. In both cases, one is interested in a series of steps that manipulate an initial state to produce a goal state.
  - 2. In essence, one way to do planning is to prove the theorem: "There exists some set of steps that will accomplish this goal".
  - 3. As we shall see, one way to construct plans is to use resolution refutation.
  - 4. There are many other approaches, of which we will look at some.
- C. Whatever form a planner may take, the problem is basically a form of states and operators search.
  - 1. Example: "blocks world"
    - a. States are different configurations of the blocks, with one being the start state. State information is expressed by descriptive predicates. We will use a somewhat simpler set than in Luger, but will also include a final state argument.
      - i. clear(X, S): block X is clear (has nothing on it) in state S
      - ii. on(X, Y, S): block X is on block Y in state S
      - iii. ontable(X, S): block X is on the table in state S
    - b. The goal state is typically not spelled out in detail - e.g. we may only care that block b is on block a, and not be concerned with where the other blocks are located - so the actual goal state is discovered by the planning process. The goal state is specified by predicates describing the things that must hold.
    - c. The operators are actions we can take to transform the state of the world - e.g. put a certain block on top of another block. Again, we will use a somewhat simpler set than in Luger:

- i. putontable(X): put block X on the table (combines either pickup or unstack with putdown - i.e. move the block from where it is now to the table)
  - ii. puton(X, Y): put block X on block Y (combines either pickup or unstack with stack - i.e. move X from where it is now be on top of Y)
2. The operators typically have associated with them a set of preconditions (what must be true in order to be able to use the operator) and postconditions (what happens as a result).

Example: The blocks world operator "put block X on block Y" has preconditions "nothing is on top of block X" and "nothing is on top of block Y" and postconditions "X is on top of Y" and "nothing is on top of X"

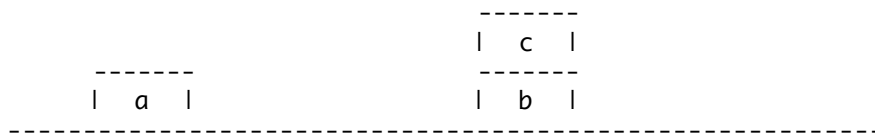
- D. While it is possible, with current technology, to construct planners for sufficiently limited domains, in general, planning remains a difficult problem, with many yet-to-be-resolved issues.

## II. Using Resolution for Planning

- A. When we introduced resolution refutation in conjunction with predicate calculus, we focused on using it to do proofs - i.e. to deal with static, factual knowledge.

Interestingly, resolution refutation can also be used to do planning!

- B. Now, suppose we had a problem like the following blocks world problem:



Develop a plan for putting c on a.

- 1. We could certainly formalize our initial state by using the "on" and "clear" predicates:

$$\text{on}(a, \text{table}) \cap \text{clear}(a) \cap \text{on}(b, \text{table}) \cap \text{on}(c, b) \cap \text{clear}(c)$$

- 2. Likewise, we could formalize our goal:  $\text{on}(c, a)$
- 3. But we couldn't solve the problem this way! How can we prove that c is on a when our axioms say it is not?
- 4. The key idea is that we have to add to our on predicate the notion of a state. Let si be the initial state and SF the final state.

a. Then we can rewrite our axioms and goal as:

$$\text{on}(a, \text{table}, \text{si}) \cap \text{clear}(a, \text{si}) \cap \text{on}(b, \text{table}, \text{si}) \cap \text{on}(c, b, \text{si}) \cap \text{clear}(c, \text{si})$$

$$(\exists \text{ SF}) \text{on}(c, a, \text{SF})$$

b. Note that  $si$  is a constant and  $SF$  is a variable. This is because  $si$  is given to us, but some of the features of  $SF$  will depend on the exact series of steps we use to accomplish our goal.

5. Further, we need some axioms to describe operators.

i. Suppose we have an operator  $puton(OBJECT1, OBJECT2, STATE)$  which can pick up any  $OBJECT1$  whose top is clear in  $STATE$  and put it on any  $OBJECT2$  whose top is clear in  $STATE$ .  $Puton$  can be defined as a function that returns a new state.

ii. We could also define an operator  $putontable(OBJECT, STATE)$  which can pick up any  $OBJECT$  whose top is clear in  $STATE$  and put it on the table. Again, it can be defined as a function returning a new state.

iii. These could be axiomized as follows. I will use  $\rightarrow$  for clarity; for actual use of the axioms we simply negate the antecedent and change the  $\rightarrow$  to  $\neg \dots \cup$

(1)  $on(X1, Y1, S1) \cap clear(X1, S1) \cap clear(Z1, S1) \rightarrow on(X1, Z1, puton(X1, Z1, S1))$

(2)  $on(X2, Y2, S2) \cap clear(X2, S2) \cap clear(Z2, S2) \rightarrow clear(Y2, puton(X2, Z2, S2))$

(3)  $on(X3, Y3, S3) \cap clear(X3, S3) \rightarrow on(X3, table, putontable(X3, S3))$

(4)  $on(X4, Y4, S4) \cap clear(X4, S4) \rightarrow clear(Y4, putontable(X4, S4))$

iv. Note that the left hand side of the axiom constitutes the PRECONDITIONS for the operator, and the right hand side gives the POSTCONDITION.

v. As an illustration of what was said about converting  $\rightarrow$ , the form of (1) that we will actually use in doing our proofs is:

$\neg[on(X1, Y1, S1) \cap clear(X1, S1) \cap clear(Z1, S1)] \cup on(X1, Z1, puton(X1, Z1, S1))$

which is equivalent to:

$\neg on(X1, Y1, S1) \cup \neg clear(X1, S1) \cup \neg clear(Z1, S1) \cup on(X1, Z1, puton(X1, Z1, S1))$

6. Now, we could start to prove out theorem, using the set of initial state axioms plus the negated goal:

(5)  $on(a, table, si)$

(6)  $clear(a, si)$

(7)  $on(b, table, si)$

(8)  $on(c, b, si)$

(9)  $clear(c, si)$

(10)  $\neg on(c, a, SF)$

i. Resolving (10) with (1) with unifier  $c/X1, a/Z1, puton(X1, Z1, S1)/SF$

(11)  $\neg on(c, Y1, S1) \cup \neg clear(c, S1) \cup \neg clear(a, S1)$

ii. Resolving (11) with (6), with unifier  $si/S1$

(12)  $\neg on(c, Y1, si) \cup \neg clear(c, si)$

iii. Resolving (12) with (9) - no unification needed

(13)  $\neg on(c, Y1, si)$

iv. Resolving (13) with (8) with unifier  $b/Y1$ :

NIL

7. We have now proved our theorem that a state does exist in which the desired condition holds. But where is our plan? It is locked up in the series of unifications:

$c/X1, a/Z1, puton(X1, Z1, S1)/SF, si/S1, b/Y1$

if we compose these by applying later operations to earlier ones, we get:  $c/X1, a/Z1, puton(c, a, si)/SF, si/S1, b/Y1$

Now our answer is the substitution for SF:  $puton(c, a, si)$

C. In our example, we saw that the answer to our problem could be found by working back through the chain of unifications to find the ultimate substitution for our initial, existentially quantified variable. Cordell Green suggested a way of automating this that has come to be called Green's device:

1. To our goal WFF we add a dummy clause  $answer(SF)$ . This will pick up the chain of unifications as they occur. (Note that  $answer(SF)$  will never participate in resolution, since there is no literal of the form  $\neg answer()$ .)

2. We terminate resolution not with NIL, but when we get down to a resolvent of the form  $answer()$ .

3. Reworking the above - axioms 1-9 are unchanged, but 10 becomes:

(10)  $\neg on(c, a, SF) \cup answer(SF)$

i. Resolving (10) with (1) with unifier  $c/X1, a/Z1, puton(X1, Z1, S)/SF$

(11)  $\neg on(c, Y1, S1) \cup \neg clear(c, S1) \cup \neg clear(a, S1) \cup answer(puton(c, a, S1))$

ii. Resolving (11) with (6), with unifier  $si/S$

(12)  $\neg on(c, Y1, si) \cup \neg clear(c, si) \cup answer(puton(c, a, si))$

iii. Resolving (12) with (9) - no unification needed

(13)  $\neg on(c, Y1, si) \cup answer(puton(c, a, si))$

iv. Resolving (13) with (8) with unifier  $b/Y1$ :

$answer(puton(c, a, si))$

D. It might appear that, at this point, we have developed a clean way to do planning. Unfortunately, we are not quite there yet. Our next example will introduce something called the frame problem.

1. Suppose our original problem were to achieve  $\text{on}(b,a)$  - a harder problem since  $C$  must first be moved out of the way. Axioms 1-9 would remain unchanged, but our negated goal (10) would become:

(10)  $\neg\text{on}(b,a,SF) \cup \text{answer}(SF)$

2. We now work through the proof, with Green's device:

- i. Resolving (10) with (1) with unifier  $b/X1, a/Z1, \text{puton}(X1,Z1,S1)/SF$

(11)  $\neg\text{on}(b,Y1,S1) \cup \neg\text{clear}(b,S1) \cup \neg\text{clear}(a,S1) \cup \text{answer}(\text{puton}(b,a,S1))$

- ii. Now we would be tempted to resolve (11) with (6), as before, using unifier  $si/S1$ :

(12)  $\neg\text{on}(b,Y1,si) \cup \neg\text{clear}(b,si) \cup \text{answer}(\text{puton}(b,a,si))$

- iii. Now the next step in the previous proof was to resolve (12) with (9). This no longer is possible, since now our last literal involves  $b$ . Further, there is no analogous resolution possible, so we move on.

- iv. One more resolution is still possible: (12) with (7) with unifier  $table/Y1$ :

$\neg\text{clear}(b,si) \cup \text{answer}(\text{puton}(b,a,si))$

- v. However, we are now stuck! Certainly, nothing resolves with the either literal. Indeed, it is not surprising that we can go no further, since the literal  $\neg\text{clear}(b,si)$  is, in fact true - so how can we hope to find a resolution that will give rise to a contradiction?

- vi. The problem, of course, is that we can no longer jump from  $si$  to  $SF$  in one move. All is not lost, though. Our original resolution of (10) with (1) was ok, but our resolution of (11) with (6) as a second step is by no means our only option at this point. Suppose, instead, we resolve (11) with (4), using unifier  $b/Y4, \text{putontable}(X4,S4)/S1$ :

(12)  $\neg\text{on}(b,Y1,\text{putontable}(X4,S4)) \cup \neg\text{clear}(a, \text{putontable}(X4,S4)) \cup \neg\text{on}(X4,b,S4) \cup \neg\text{clear}(X4,S4) \cup \text{answer}(\text{puton}(b,a,\text{putontable}(X4,S4)))$

- vii. Now we can resolve (12) with (8) with unifier  $c/X4, si/S4$ :

(13)  $\neg\text{on}(b,Y4,\text{putontable}(c,si)) \cup \neg\text{clear}(a, \text{putontable}(c,si)) \cup \neg\text{clear}(c,si) \cup \text{answer}(\text{puton}(b,a,\text{putontable}(c,si)))$

- viii. This now resolves with (9) directly:

(14)  $\neg\text{on}(b,Y4,\text{putontable}(c,si)) \cup \neg\text{clear}(a, \text{putontable}(c,si)) \cup \text{answer}(\text{puton}(b,a,\text{putontable}(c,si)))$

- ix. Unfortunately, we're stuck again! To finish our proof, we want to resolve with (6) and (7). But 6 and 7 have the constant state `si`, and (14) has in each case `putontable(c,si)`, which does not unify. However, unlike our previous hangup, this one does not involve a logical contradiction. Our goal is to resolve down to a contradiction, and (14) is patently false, since putting `C` on the table in no way changes the fact that `B` is originally on something (the table) and `A` is originally clear.
- E. The reason why our last proof failed is because our original axioms describing the operators were not complete. They told how the operators affected the blocks that were involved in the operation, but did not specify how the operators affected the other blocks - no effect at all.
1. Evidently, we need axioms that tell us that if I put a block on the table, then the position of any other block is unchanged. Such axioms are called frame axioms. In our case, we need the following:
    - (F1) `on(VF1,WF1,SF1) ∧ different(VF1,XF1) → on(VF1,WF1,puton(XF1,ZF1,SF1))`
    - (F2) `clear(VF2,SF2) ∧ different(VF2,ZF2) → clear(VR2,puton(XF2,ZF2,SF2))`
    - (F3) `on(VF3,WF3,SF3) ∧ different(VF3,XF3) → on(VF3,WF3,putontable(XF3,SF3))`
    - (F4) `clear(VF4,SF4) → clear(VF4,putontable(XF4,SF4))`  
 -- why no different here? collapsing two cases: the object put on the table was clear before and remains clear after, and any other object which was clear before remains clear.
  2. We also need some way of handling different - either a set of axioms or making different a built-in function. For this example, we will use axioms like:
    - (D1) `different(a,b)`
    - (D2) `different(b,c)`
    - ...
  3. Now we can finish our proof:
    - i. Resolving (14) with (F4), with unifier `a/VF4, c/XF4, si/SF4`:
      - (15) `¬on(b,Y4,putontable(c,si)) ∨ ¬clear(a,si) ∨ answer(puton(b,a,putontable(c,si)))`
    - ii. Resolving (15) with (F3), with unifier `b/VF3, Y4/WF3, c/XF3, si/SF3`:
      - (16) `¬clear(a,si) ∨ ¬on(b,Y4,si) ∨ ¬different(b,c) ∨ answer(puton(b,a,putontable(c,si)))`
    - iii. Resolving (16) with (6) with unifier `table/Y4`
- (17) `¬clear(a,si) ∨ ¬different(b,c) ∨ answer(puton(b,a,putontable(c,si)))`

iv. Resolving (17) with (7)

(18)  $\neg\text{different}(b,c) \cup \text{answer}(\text{puton}(b,a,\text{putontable}(c,\text{si})))$

v. Finally, resolving (18) with (D2), we get

$\text{answer}(\text{puton}(b,a,\text{putontable}(c,\text{si})))$

vi. Our plan is therefore (reading outward from si):

put C on the table  
put B on A

F. An example of this in action: hand out and discuss BLOCKS.PRO

### III. STRIPS

--- -----

- A. We now look at another approach to planning called STRIPS. The original system was developed in 1971, and has been the basis of a number of systems since then.
- B. In contrast to the theorem-proving planning approach we just looked at, in a STRIPS system the predicates describing the world do not have a state attribute. Instead, the database itself is altered to reflect what is happening in the plan.
- C. Each strips operator has three lists associated with it:
1. Preconditions - clauses that must be true in the world in order for the operator to be capable of being applied.
  2. Add list - clauses which become true as a result of the operator being applied.
  3. Delete list - clauses that are no longer true after the operator has been applied.
  4. Example: our basic blocks world operations could be described by STRIPS operators as follows:

$\text{puton}(X,Y):$

Preconditions:  $\text{on}(X, Z) \cap \text{clear}(X) \cap \text{clear}(Y)$

Add:  $\text{on}(X, Y), \text{clear}(Z)$

Delete:  $\text{on}(X, Z), \text{clear}(Y)$

where X and Y are variables standing for the block to be moved and the block it is put on, and Z stands for some other thing (another block or the table) that X is on before it is moved. (This is listed as a precondition so that the correct fact can be deleted when it is moved).

putontable(X):

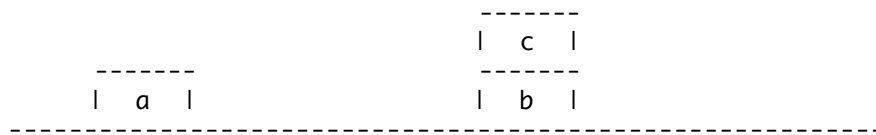
Preconditions:  $\text{on}(X, Y) \cap \text{clear}(X)$

Add:  $\text{on}(X, \text{table}), \text{clear}(Y)$

Delete:  $\text{on}(X, Y)$

5. STRIPS addresses the frame problem by the STRIPS ASSUMPTION: any clause that is true before an operator is applied, and is not on the operator's delete list, remains true after it is applied. (Contrast this with the frame axiom approach we have just used, which requires explicit axioms about what remains true.)

- D. An example: the STRIPS version of the second problem we solved by resolution:



Develop a plan for putting b on a.

1. Initial world:  $\text{on}(a, \text{table}).$   
 $\text{on}(b, \text{table}).$   
 $\text{on}(c, b).$   
 $\text{clear}(a).$   
 $\text{clear}(c).$   
 $\text{clear}(\text{table}).$  /\* Always room for one more \*/

Goal:  $\text{on}(b, a).$

2. The preconditions for putontable(X) are satisfied with  $X = c$  and  $Y = b$ . Applying this operator deletes  $\text{on}(c, b)$  and adds  $\text{on}(c, \text{table})$  and  $\text{clear}(b)$ , leading to the new world:

```

on(a, table).
on(b, table).
on(c, table).
clear(a).
clear(b).
clear(c).
clear(table).
```

3. Now the preconditions for puton(X, Y) are satisfied with  $X = b$  and  $Y = a$ . Applying this operator deletes  $\text{on}(b, \text{table})$  and  $\text{clear}(a)$ , and adds  $\text{on}(b, a)$  and  $\text{clear}(\text{table})$  [ which is already true, so not added again].

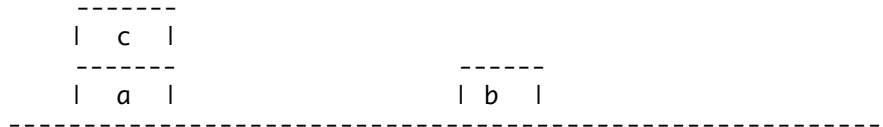
```

on(a, table)
on(b, a)
on(c, table)
clear(b)
clear(c)
clear(table).
```

- E. Clearly, except for trivial goals, constructing a plan involves a search through the space of operators and resulting world descriptions - a question we begged in the previous example. This can be done using either forward or backward reasoning.
1. If forward reasoning is used, some heuristic measure of closeness to the goal must be used to prevent combinatorial explosion.
  2. An interesting variant of STRIPS - called recursive STRIPS - uses the following backward chaining approach:
    - a. Find an operator whose add list contains an element that unifies with the goal we are trying to satisfy.
    - b. Work backward from the add list to the preconditions that must hold to make the operator applicable. Treat any of these preconditions that does not hold in the starting world as a new goal - stopping when all components of the revised goal are satisfied by the initial world.
    - c. Example: the problem we just solved
      - i. The only operator having an add list item that unifies with the goal is `puton(X,Y)` with  $X = b$ ,  $Y = a$  - which becomes `puton(b, a)`. The preconditions for this operator (after substituting variables) are
 
$$\text{on}(b, Z) \cap \text{clear}(b) \cap \text{clear}(a)$$

Two of these are satisfied by the original world ( `on(b, Z)` with substitution  $Z = \text{table}$ ; `clear(a)` ).
      - ii. We call STRIPS recursively with `clear(b)` as the goal. Here, `putontable(X)` has an add list item that unifies with this goal (with substitution  $Y = b$ ). The preconditions for this operator (after substituting variables) are
 
$$\text{on}(X, b) \cap \text{clear}(X).$$

The first is satisfied by the original world with substitution  $X = c$ , and `clear(c)` also holds in the original world, so we are done (and we can conclude that the `putontable` operation we need is `putontable(c)`).
- F. STRIPS planning systems use various means to break a larger problem up into smaller pieces. Some of these are discussed in the book - we consider just one here, because it leads to an interesting problem.
1. Frequently, the goal is a conjunction of clauses - in which case the planner can attempt to achieve them one at a time. Of course, there is always the danger that achieving a later clause in the goal may undo an earlier one.
  2. A particularly difficult situation that can arise from compound initial goals is known as the SUSSMAN ANOMALY - of which the following problem is an illustration:



Goal:  $\text{on}(a, b) \cap \text{on}(b, c)$

- a. We can achieve the first conjunct by putting C on the table, then A on B. But then, in order to achieve the second conjunct, we have to take A off B!
- b. Alternately, we could achieve the second conjunct first by `puton(b, c)`. But now we have to take it back off in order to achieve the first conjunct.
- c. Clearly, there is no way to break a problem like this into two separate, independently solvable problems.
- d. Does the resolution based blocks world Prolog planner have this problem?

ASK

DEMO: BLOCKS.PRO - `put z on w and put x on z.`

`put x on z and put z on w.`