10 Planning in AI Agent World

Syllabus

The Blocks World, Components of a Planning System, Goal Stack Planning, Nonlinear Planning Using Constraint Posting, Hierarchical Planning, Reactive Systems, Other Planning Techniques.

Contents

10.1 Blocks World

GTU : Summer-16

- Inorder to compare the variety of methods of planning, we should find it useful to look at all of them in a single domain that is complex enough that the need for each of the mechanisms is apparent yet simple enough that easy-to follow examples can be found.
- There is a flat surface on which blocks can be placed.
- There are a number of square blocks, all of the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks

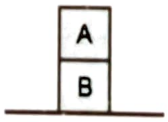
Why Use the Blocks world as an example ?

The blocks world is chosen because :

- It is sufficiently simple and well behaved.
- It is easily understood
- Yet still provides a good sample environment to study planning :
  - Problems can be broken into nearly distinct subproblems.
  - We can show how partial solutions need to be combined to form a realistic complete solution.

Actions of the robot arm

- UNSTACK(A,B)



- STACK(A,B)
- PICKUP(A)
- PUTDOWN(A)
- Notice that the robot arm can hold only one block at a time.

Predicates

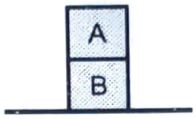
Inorder to specify both the conditions under which an operation may be performed and the results of performing it, we need the following predicates :

- ON(A,B)
- ONTABLES(A)
- CLEAR(A)

- HOLDING(A)
- ARMEMPTY

Simple Position

- State S0



- ON(A, B, S0) ^ ONTABLE(B, S0)^CLEAR(A, S0)

If we execute UNSTACK(A,B) in state S0, then the resulting state S1 : HOLDING(A,S1)^CLEAR(B,S1).

To enable the complete situation to be described, we provide a set of rules called **frame axioms**, that describe components of the state that are not affected by each operator.

- ONTABLE(x,s) -> ONTABLE(z, DO(UNSTACK(x,y),s))
- DO is a function that specifies for a given state and a given action, the new state that results from the execution of the action.

Robot - problem solving system (STRIPS)

- List of new predicates that the operator causes : ADD, DELETE
- PRECONDITIONS list contains those predicates that must be true for the operator to be applied.

STRIPS style operators for BLOCKs World

- STACK(x,y)
  - P : CLEAR(y)^HOLDING(x)
  - D : CLEAR(y)^HOLDING(x)
  - A : ARMEMPTY^ON(x,y)
- PICKUP(x)
  - P : CLEAR(x) ^ ONTABLE(x) ^ARMEMPTY
  - D : ONTABLE(x) ^ ARMEMPTY
  - A : HOLDING(x)

A simple search tree

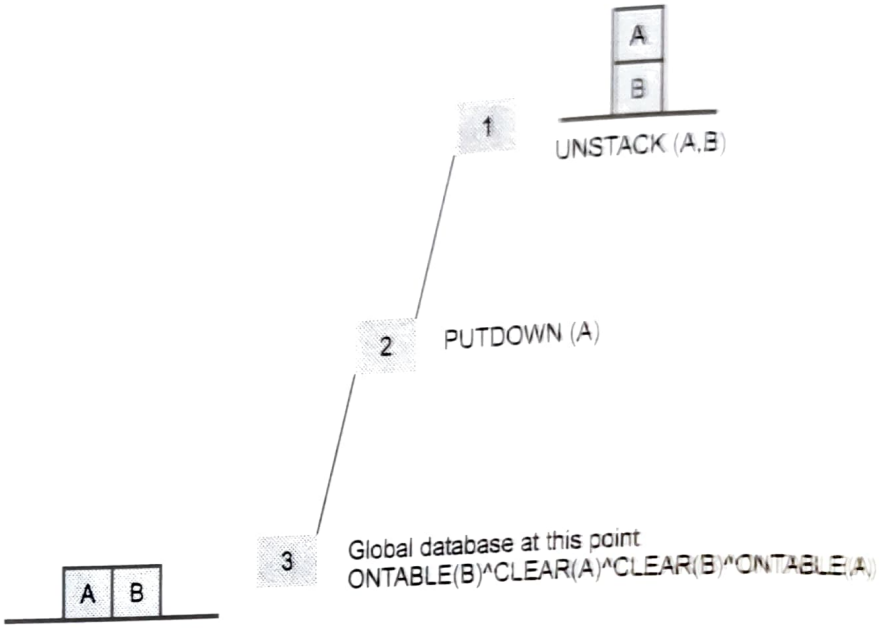


Fig. 10.1.1

10.2 Planning

GTU : Summer-14

The methods which focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interactions among the subparts as they are detected during the problem-solving process are often called as planning. Planning refers to the process of computing several steps of a problem-solving procedure before executing any of them.

What does planning involve ?

Planning problems are hard problems :

- They are certainly non-trivial.
- Solutions involve many aspects that we have studied so far :
  - Search and problem solving strategies.
  - Knowledge representation schemes.
  - Problem decomposition - breaking problem into smaller pieces and trying to solve these first.

We have seen that it is possible to solve a problem by considering the appropriate form of knowledge representation and using algorithms to solve parts of the problem and also to use searching methods.

Search in planning

Search basically involved moving from an *initial state* to a *goal state*. Classic search techniques could be applied to planning state in this manner :

A* Algorithm

- Best first search,

Problem decomposition

- *Synthesis,* Frame Problem.

AO* Algorithm

- Split problem into *distinct* parts.

Heuristic reasoning

Ordinary search backtracking can be hard so introduce reasoning to devise heuristics and to control the backtracking.

The first major method considered the solution as a search from the initial state to a goal state through a state space. There are many ways of moving through this space by using operators and the *A* algorithm* described the best first search through a graph. This method is fine for simpler problems but for more realistic problems it is advisable to use problem decomposition. Here the problem is split into smaller sub problems and the partial solutions are then *synthesised*. The **danger** in this method occurs when certain paths become *abortive* and have to be discarded. How much of the partial solution can be saved and how much needs to be recomputed.

The *frame problem* - deciding which things change and which do not - gave some guidance on enabling us to decide on what stays the same and on what changes as we go from state to state. If the problem concerned designing a robot to build a car then mounting the engine on the chassis would not affect the rear of the car now-a-days.

The *AO* algorithm* enabled us to handle the solution of problems where the problem could be split into distinct parts and then the partial solutions reassembled. However difficulties arise if parts interacted with one another. Most problems have some interaction and this implies some thought into the ordering of the steps; for example if the robot has to move a desk with objects on it from one room to another; or to move a sofa from one room to another and the piano is near the doorway. The thought process involved in recombining partial solutions of such problems is known as planning. At this point we run into a discussion about the role of the computer in the design of a plan as to how we can solve a problem. It is extremely unlikely at this stage that a computer will actually solve the problem unless it is a game and here some interaction with a person is needed.

Generally the computer is used to decide upon or to offer words of wisdom on the best method of approaching the solution of a problem. In one sense this can be interpreted as a simulation; if we are considering the handling of queues at an airport or a post office there are no actual people and so we can try a range of possibilities; likewise in a traffic control problem there are no cars or aircraft piling up at a junction or two miles over an airport. Once the computer based investigation has come up with the best solution we can then implement it at the real site.

This approach assumes that there is continuity in the way of life. It cannot budget for rapid change or revolution. How can this approach cater for unexpected events such as a faulty component or a spurious happening such as two items stuck together or a breakdown in the path somewhere vital such as in a camera reel. When a fault or some unrecognizable state is encountered it is not necessary to restart for much of what has been successfully solved is still useful. Consider a child removing the needles from a partially completed knitted sweater. The problem lies in restarting from a dead end and this will need some backtracking. This method of solution is pursued to reduce the level of complexity and so to ensure successful handling we must introduce reasoning to help in the backtracking required to cater for faults. To assist in controlling backtracking many methods go backward from the goal to an initial state.

- **Components of a planning system**

1. Choose the best rule to apply next based on the best available heuristic information.
2. Apply the chosen rule to compute the new problem state that arises from its application.
3. Detect when a solution has been found.
4. Detect when an almost correct solution has been found and employ special techniques to make it totally correct.
5. Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.

1. Choose the rules to apply

- The most widely used technique for selecting an appropriate rules to apply is first to isolate a set of differences between desired goal state and then to identify those rules that are relevant to reducing those differences.
- If several rules, a variety of other heuristic information can be exploited to choose among them.

For example, if we wish to travel by car to visit a friend, then

- The first thing to do is to fill up the car with fuel.

- If we do not have a car then we need to acquire one.
- The largest difference must be tackled first.

2. Applying rules

- In simple systems, applying rules is easy. Each rule simply specified the problem state that would result from its application.
- In complex systems, we must be able to deal with rules that specify only a small part of the complete problem state.
- One way is to describe, for each action, each of the changes it makes to the state description.
- A number of approaches to this task have been used.

Green's Approach (1969)

Basically this states that *we note the changes to a state produced by the application of a rule.* Consider the problem of having two blocks *A* and *B* stacked on each other (*A* on top). Then we may have an initial state S_0 which could be described as :

$$ON(A, B, S_0) \wedge$$
$$ONTABLE(B, S_0) \wedge$$
$$CLEAR(A, S_0)$$

If we now wish to UNSTACK(*A*, *B*). We express the operation as follows :

$$[CLEAR(x,s) \wedge ON(x, y, s)] \rightarrow$$
$$[HOLDING(x, DO(UNSTACK(x,y),s)) \wedge$$
$$CLEAR(y,DO(UNSTACK(x,y),s))]$$

where x, y are any blocks, s is any state and DO() specifies that a new state results from the given action.

The result of applying this to state S_0 to give state S_1 then we get :

$HOLDING(A, S_1)$ $CLEAR(B, S_1)$.

There are a few problems with this approach :

The frame problem

- In the above we know that *B* is still on the table. This needs to be encoded into *frame axioms* that describe components of the state *not* affected by the operator.

The qualification problem

- If we resolve the frame problem the resulting description may still be inadequate. Do we need to encode that a block cannot be placed on top of itself ? If so should this attempt *fail* ?

If we allow failure things can get complex - do we allow for a lot of unlikely events ?

The ramification problem

- After unstacking block *A*, previously, how do we know that *A* is no longer at its initial location ? *Not only is it hard to specify exactly what does not happen (frame problem) it is hard to specify exactly what does happen.*

State Description Approach

In this approach for each action, a change it makes to the state is described, everything else remains unchanged.

The main **advantage** of this method is single mechanism (that is resolution) can perform all operations that are required for state description. The major **disadvantage** is the number of axioms that are required becomes very large if problem-state description is complex.

STRIPS Approach (1971)

STIPS proposed another approach :

- Basically each operator has three lists of predicates associated with it :
  - a list of things that become TRUE called ADD.
  - a list of things that become FALSE called DELETE.
  - a set of prerequisites that must be true before the operator can be applied.
- Anything not in these lists is assumed to be unaffected by the operation.
- This method initial implementation of STRIPS - has been *extended* to include other forms of reasoning/planning (*e.g.* Nonmonotonic methods, Goal Stack Planning and even Nonlinear planning - We will see later.)

Consider the following example in the Blocks World and the fundamental operations :

STACK

- Requires the arm to be holding a block A and the other block B to be clear. Afterwards the block A is on block B and the arm is empty and these are true - ADD; The arm is not holding a block and block B is not clear; predicates that are false DELETE;

UNSTACK

- Requires that the block A is on block B; that the arm is empty and that block A is clear. Afterwards block B is clear and the arm is holding block A - ADD; The arm is not empty and the block A is not on block B - DELETE;

We have now greatly reduced the information that needs to be held. If a new attribute is introduced we do not need to add new axioms for existing operators. Unlike in Green's method we remove the state indicator and use a database of predicates to indicate the current state

Thus if the last state was :

ONTABLE(B) ^ ON(A,B) ^ CLEAR(A)

after the unstack operation the new state is

ONTABLE(B) ^ CLEAR(B) ^ HOLDING(A) ^ CLEAR(A)

3. Detecting Progress

The final solution can be detected if,

- We can devise a predicate that is true when the solution is found and is false otherwise.
- Requires a great deal of thought and requires a proof.

4. Detecting a solution

- A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.
- How will it know when this has been done ?
- In simple problem-solving systems, this question is easily answered by a straightforward match of the state descriptions.
- One of the representative systems for planning systems is, predicate logic. Suppose that as a part of our goal, we have the predicate P(x). To see whether P(x) is satisfied in some state, we ask whether we can prove P(x) given the assertions that describe that state and the axioms that define the world model.

5. Detecting Dead Ends

- As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.
- The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.
- If the search process is **reasoning forward** from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.
- If search process is **reasoning backward** from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or because little progress is being made. In backward reasoning each goal is decomposed into subgoals. Each of these subgoals may lead to a set of additional subgoals. The major disadvantage is that every subgoal can lead to multiple subgoals making a problem harder than original one.

Detecting false trails is also necessary :

- E.g. A* search - if insufficient progress is made then this trail is aborted in favour of a more hopeful one.
- Sometimes it is clear that solving a problem one way has reduced the problem to parts that are harder than the original state.
- By moving back from the goal state to the initial state it is possible to detect conflicts and any trail or path that involves a conflict can be pruned out.
- Reducing the number of possible paths means that there are more resources available for those left.

Supposing that the computer teacher is ill at a school there are two possible alternatives :

- Ttransfer a teacher from mathematics who knows computing or
- Bring another one in.

Possible Problems :

- If the mathematics teacher is the only teacher of mathematics the problem is not solved.
- If there is no money left the second solution could be impossible.

If the problems are nearly decomposable we can treat them as decomposable and then patch them, how ? Consider the final state reached by treating the problem as decomposable at the current state and noting the differences between the goal state and the current state, and the goal state and the initial state and use appropriate Means End analysis techniques to move in the best direction. Better is to work back in the path leading to the current state and see if there are options. It may be that one optional path could lead to a solution whereas the existing route led to a conflict. Generally this means that some conditions are changed prior to taking an optional path through the problem.

Another approach involves putting off decisions until one has to, leaving decision making until more information is available and other routes have been explored. Often some decisions need not be taken as these nodes are never reached.

Repairing an Almost Correct Solution

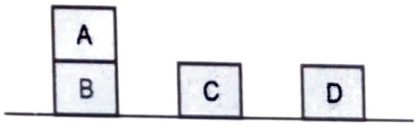
- The kinds of techniques we are discussing are often useful in solving nearly decomposable problems.
- One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the sub-problems separately, and then check that when the sub-solutions are combined, they do in fact yield a solution to the original problem.

10.3 Goal Stack Planning

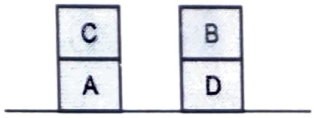
- In this method, the problem solver makes use of a single stack that contains both goals and operators that have been proposed to satisfy those goals.
- The problem solver also relies on a database that describes the current situation and a set of operators described as PRECONDITION, ADD, and DELETE lists.
- The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order.
- A plan generated by this method contains a sequence of operators for attaining the first goal, followed by a complete sequence for the second goal etc.
- At each succeeding step of the problem solving process, the top goal on the stack will be pursued.
- When a sequence of operators that satisfies it is found, that sequence is applied to the state description, yielding new description.
- Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal.
- This process continues until the goal stack is empty.
- Then as one last check, the original goal is compared to the final state derived from the application of the chosen operators.
- If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted onto the stack and the process is resumed.

Goal stack planning

- To start with goal stack is simply :
- On (C, A)^ON (B, D)^ONTABLE(A)^ONTABLE(D)



Start :
ON(B, A)^ONTABLE(A)^
ONTABLE(C)
^ONTABLE(D)
^ARMEMPTY



Goal : ON(C, A)^ON(B, D)^
ONTABLE(A)^ONTABLE(D)

- This problem is separate into four subproblems, one for each component of the goal.
- Two of the subproblems ONTABLE(A) and ONTABLE(D) are already true in the initial state.
- Alternative 1: Goal stack :
- ON (C, A)
- ON (B, D)
- ON (C, A)^ON(B, D)^OTAD
- Alternative 2 : Goal stack :
- ON (B, D)
- ON (C,A)
- ON (C,A)^ON(B,D)^OTAD

***Exploring Operators**

- Pursuing alternative 1, we check for operators that could cause ON(C,A)
- Of the 4 operators, there is only one STACK. So it yields :
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Preconditions for STACK(C,A) should be satisfied, we must establish them as subgoals :
- **CLEAR(A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Here we exploit the Heuristic that if HOLDING is one of the several goals to be achieved at once, it should be tackled last.
- Next we see if CLEAR(A) is true, it is not. The only operator that could make it true is UNSTACK(B, A). This produces the goal stack :
- **ON(B,A)**
- **CLEAR(B)**
- **ON (B,A)^CLEAR(B)^ARMEMPTY**

- **UNSTACK(B,A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- We see that we can pop predicates on the stack till we reach HOLDING(C) for which we need to find suitable operator.
- The operators that might make HOLDING(C) true : PICKUP(C) and UNSTACK(C,x). Without looking ahead, since we cannot tell which of these operators is appropriate, we create two branches of the search tree corresponding to the following goal stacks :

ALT1 :	ALT2 :
ONTABLE(C)	ON(C,X)
CLEAR(C)	CLEAR(C)
ARMEMPTY	ARMEMPTY
ONTABLE(C) ^CLEAR(C)^ARMEMPTY	ON(C,X)^CLEAR(C)^ARMEMPTY
PICKUP(C)	UNSTACK(C,X)
CLEAR(A)^HOLDING (C)	CLEAR(A)^HOLDING (C)
STACK(C, A)	STACK(C,A)
ON(B, D)	ON(B,D)
ON(C,A)^ON(B,D)^OT AD	ON(C,A)^ON(B,D)^OT AD

Table 10.3.1 Goal stack

***Choosing Alternative**

- How should our program choose now between alternative 1 and alternative 2 ?
- We can tell that picking up C(alt 1) is better than unstacking it because it is not currently on anything. So to unstack it, we would first have to stack it. This would be waste of effort.
- But how could the program know that ?
- If we pursue the alternative 1, the top element on the goal stack is ONTABLE(C) which is already satisfied, so we pop it off. CLEAR(C) is also satisfied and is popped off.

- The remaining precondition of PICKUP(C) is ARMEMPTY which is not satisfied since HOLDING(B) is true.So we apply the operator STACK(B,D). This makes the goal stack :
- **CLEAR(D)**
- **HOLDING(B)**
- **CLEAR(D)^HOLDING(B)**
- **STACK(B,D)**
- **ONTABLE(C)^CLEAR(C)^ARMEMPTY**
- **PICKUP(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**

***Complete plan**

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A,B)
5. UNSTACK(A,B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B,C)
9. PICKUP(A)
10. STAKC(A,B)

10.4 Non-linear Planning using Constraint Posting

GTU : Summer-18

- Difficult problems cause goal interactions,
- The operators used to solve one subproblem may interfere with the solution to a previous subproblem.
- Most problems require an interwined plan in which multiple subproblems are worked on simultaneously.
- Such a plan is called nonlinear plan because it is not composed of a linear sequence of complete subplans.

- Let us reconsider the **SUSSMAN ANOMALY.**
- Problems such as this one require subproblems to be worked on simultaneously.
- Thus a nonlinear plan using heuristics such as :

1. Try to achieve ON(A,B) clearing block A putting block C on the table.
2. Achieve ON(B,C) by stacking block B on block C.
3. Complete ON(A,B) by stacking block A on block B.

Constraint posting has emerged as a central technique in recent planning systems (e.g. MOLGEN and TWEAK)

Constraint posting builds up a plan by :

- Suggesting operators,
- Trying to order them and
- Produce bindings between variables in the operators and actual blocks.

The initial plan consists of *no* steps and by studying the goal state ideas for the possible steps are generated. There is *no order or detail* at this stage. Gradually more detail is introduced and constraints about the order of subsets of the steps are introduced until a *completely ordered* sequence is created. In this problem means-end analysis suggests two steps with end conditions ON(A,B) and ON(B,C) which indicates the operator STACK giving the layout shown below where the operator is preceded by its preconditions and followed by its post conditions :

CLEAR(B)	CLEAR(C)
*HOLDING(A)	*HOLDING(B)
STACK(A,B)	STACK(B,C)
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
CLEAR(B)	CLEAR(C)
¬ HOLDING(A)	¬ HOLDING(B)

NOTE

- There is no order at this stage.
- Unachieved preconditions are starred (*).
- Both of the HOLDING preconditions are unachieved since the arm holds nothing in the initial state.
- Delete postconditions are marked by (¬).

Many planning methods have introduced heuristics to achieve goals or preconditions. The TWEAK planning method brought all these together under one formalism. Other methods that introduced/used the following heuristics are mentioned in brackets in the following section.

***Constraint Posting**

- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators, partial orderings between operators, and binding of variables within operators.
- At any given time in the problem-solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should be ordered with respect to each other.
- A solution is a partially ordered, partially instantiated set of operators to generate an actual plan, we convert the partial order into any of a number of total orders.

***Constraint posting versus state space search**

State Space Search

- Moves in the space :

  Modify world state via operator
- Model of time :

  Depth of node in search space
- Plan stored in :

  Series of state transitions

Constraint Posting Search

- Moves in the space :
  - Add operators
  - Order operators
  - Bind variables
  - Or Otherwise constrain plan
- Model of Time :
  - Partially ordered set of operators
- Plan stored in :
  - Single node

The *Tweak* planning method involves the following heuristics :

Step Addition

- Creating new steps (GPS).

Promotion

- Constraining a step to go before another step (Sussman's HACKER).

Declobbering

- Placing a new step between two steps to revert a precondition (NOAH, NONLIN).

Simple Establishment

- Assigning a value to a variable to ensure a precondition (TWEAK).

Separation

- Preventing variables being assigned certain values (TWEAK).

Now lets look at the effect of each heuristic.

We must try and achieve the preconditions of the STACK operation above. We could try picking up the respective blocks :

CLEAR(A)	CLEAR(C)
ONTABLE(A)	ONTABLE(B)
*ARMEMPTY	*ARMEMPTY
PICKUP(A)	PICKUP(B)
ONTABLE(A)	¬ ONTABLE(B)
¬ ARMEMPTY	¬ ARMEMPTY
HOLDING(A)	HOLDING(B)

At the moment there is no plan as the postconditions of this set could negate the preconditions of the first (STACK) plan so we must introduce order :

- If the eventual plan contains a PICKUP then STACK step then.
- HOLDING preconditions would need to be satisfied by some other steps.
- *Solve* this by enforcing ordering by introducing constraints whenever step addition is employed.
- In this case we need to state that a PICKUP step should precede a corresponding STACK step. That is to say

  PICKUP(A) ← STACK(A,B)

  PICKUP(B) ← STACK(B,C)

This gives four steps partially ordered and four unachieved conditions.

- *CLEAR(A) - Block A is not clear in initial state.
- *CLEAR(B) - Although block B is clear in initial state STACK(A,B) with postcondition CLEAR(B) might precede the step with *CLEAR(B) precondition.
- Two *ARMEMPTY - initial state makes ARMEMPTY but PICKUP step has ARMEMPTY and could again precede this step.

We can use the *promotion* heuristic to force one operator to precede another so that the postcondition of one operator STACK(A,B) does not negate the precondition CLEAR(B) of another operator PICKUP(B). This ordering is represented by

PICKUP(B) ← STACK(A,B)

We can use promotion to achieve one of the ARMEMPTY preconditions :

Making PICKUP(B) precede PICKUP(A) ensures that the arm is empty and all the conditions for PICKUP(B) are met.

This is written

PICKUP(B) ← PICKUP(A).

Unfortunately a postcondition of the first operator is that the arm becomes not empty, so we need to use the *declobbering* heuristic to achieve the preconditions of the second operator PICKUP(A).

Declobbering

- PICKUP(B) asserts ¬ ARMEMPTY.
- But if we insert a step between PICKUP(B) and PICKUP(A) to reassert ARMEMPTY then we can achieve the precondition.
- STACK(B,C) can do this so we *post another constraint :*
  PICKUP(B) ← STACK(B,C) ← PICKUP(A)

We still need to achieve CLEAR(A) :

The appropriate operator is UNSTACK(x,A) by *step addition*. This leads to the following set of conditions

***CLEAR(x)**
***ON(x,A)**
***ARMEMPTY**

UNSTACK(x,A)

ON(x,A)
¬ ARMEMPTY
HOLDING(x)
CLEAR(A)

The variable x can be bound to the block C by the *simple establishment* heuristic since C is on A in the initial state. The preconditions CLEAR(C) and ARMEMPTY are negated by STACK(B,C) and by PICKUP(B) or PICKUP(A) however.

So we must introduce three orderings by *promotion* to ensure the operator UNSTACK(C,A).

UNSTACK(C,A) ***leftarrow*** **STACK(B,C)**

UNSTACK(C,A) ***leftarrow*** **PICKUP(A)**

UNSTACK(C,A) ***leftarrow*** **PICKUP(B)**

Promotion involves adding a step and this clobbers one of the preconditions of PICKUP(B) viz ARMEMPTY, always a potential problem with this heuristic.

However all is not lost as there is an operator, PUTDOWN that has the required postcondition and given that the operator UNSTACK(C,A) had generated the precondition for it of HOLDING(C) we can produce an extra operator successfully

HOLDING(C)

PUTDOWN(C)

¬ HOLDING(C)

ONTABLE(C)

ARMEMPTY

This operator *declobbers* the operator PICKUP(B) yielding the sequence

UNSTACK(C,A) ← PUTDOWN(C) ← PICKUP(B)

This yields the final sequence :

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK (B,C)
5. PICKUP(A)
6. STACK(A,B)

Let us finish this section by looking at the formal form of the TWEAK algorithm :

1. Initialize S to be the set of propositions in the goal state.
2. Repeat
   a. Remove some unachieved proposition P from S.
   b. Achieve P by using one of the heuristics.
   c. Review all the steps, including additional steps to find all unachieved preconditions, add these to S the set of unachieved preconditions until the set S is empty.
3. Complete the plan by converting partial orders into a total order performing all necessary instantiations, binding of the variables.

*Modal Truth Criterion

- A proposition P is necessarily true in a state S if and only if two conditions hold: there is a state T equal or necessarily previous to S in which P is necessarily asserted; and for every step C possibly before S and every proposition Q possibly co-designating with P which C denies, there is a step W necessarily between C and S which asserts R, a proposition such that R and P co-designate whenever P and Q co-designate.
- Modal truth criterion tells us when a proposition is true.

10.5 Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans.
- It is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found.
- Then an attempt can be made to fill the appropriate details.
- Early attempts to do this involved the use of macro operators.
- But in this approach, no details were eliminated from actual descriptions of the operators.
- As an example, suppose you want to visit a friend in Europe but you have a limited amount of cash to spend. First preference will be find the airfares, since finding an affordable flight will be the most difficult part of the task. You should not worry about getting out of your driveway, planning a route to the airport etc, until you are sure you have a flight.
- The assignment of appropriate criticality value is critical to the success of this hierarchical planning method.
- Those preconditions that no operator can satisfy are clearly the most critical.

- Example, solving a problem of moving robot, for applying an operator, PUSH-THROUGH DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is nothing we can do about it if it is not true.

ABSTRIPS

- ABSTRIPS actually planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction were ignored.
- ABSTRIPS approach is as follows :
  - First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
  - These values reflect the expected difficulty of satisfying the precondition.
  - To do this, do exactly what STRIPS did, but simply ignore the preconditions of lower than peak criticality.
  - Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next-lowest criticality level.
  - Because this approach explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has been called **length-first approach.**

10.6 Reactive Systems

- The idea of reactive systems is to avoid planning altogether, and instead use the observable situation as a clue to which one can simply react.
- A reactive system must have access to a knowledge base of some sort that describes what actions should be taken under what circumstances.
- A reactive system is very different from the other kinds of planning systems we have discussed because it chooses actions one at a time.
- It does not anticipate and select an entire action sequence before it does the first thing.
- Example is a thermostat. The job of the thermostat is to keep the temperature constant.
- Reactive systems are capable of surprisingly complex behaviours.
- The main advantage reactive systems have over traditional planners is that they operate robustly in domains that are difficult to model completely and accurately.
- Reactive systems dispense with modeling altogether and base their actions directly on their perception of the world.

- Another advantage of reactive systems is that they are extremely responsive, since they avoid the combinatorial explosion involved in deliberative planning.
- This makes them attractive for real time tasks such as driving and walking.

Thermostat

- Is an example for reactive systems.
- Its job is to keep the temperature constant inside a room.
- One might imagine a solution to this problem that requires significant amounts of planning, taking into account how the external temperature rises and falls during the day, how heat flows from room to room, and so forth.
- Real thermostat uses simple pair of situation-action rules :
  1. If the temperature in the room is k degrees above the desired temperature, then turn the airconditioner on.
  2. If the temperature in the room is k degrees below desired temperature, then turn the airconditioner off.

10.7 Other Planning Techniques

- Triangle tables [Fikes et al.,1972] - It provides a way recording the goals that each operator is expected to satisfy as well as the goals that must be true for its correct execution. If some exceptional situation occurs during execution of the plan, then a table is provided that stores the information which is required to patch the plan.
- Metaplanning [Stefik,1981 a] - It is a technique for reasoning about the problem being solved, which also provides a way organizing the planning process.
- Macro-operators [Fikes and Nilsson, 1971] - It allows a planner to build new operators which represent commonly used sequences of operators.
- Case-based planning [Hammond, 1986] - It makes use of already existing plan to develop new plans there by providing reusability of existing plans.

Answer in Brief

1. *Explain various planning techniques ?* **(Refer sections 10.2, 10.3, 10.4 and 10.5)**
2. *What is regression ?* **(Refer section 10.2)**
3. *What are differences and similarities between problem solving and planning ?* **(Refer section 10.2)**
4. *Explain the concept of planning with state-space search with suitable example.* **(Refer section 10.2)**
5. *Write algorithms for TWEAK and STRIPS.* **(Refer section 10.2)**
6. *Explain goal stack planning method.* **(Refer section 10.3)**

7. *Explain various ways of applying rules in complex system.* **(Refer section 10.4)**
8. *Explain non-linear planning method.* **(Refer section 10.4)**
9. *Explain heirarchical planning with example.* **(Refer section 10.5)**
10. *Write short note on learning methods.* **(Refer section 10.2)**
11. *Design the operators and specify their respective precondition (P), delete (D) and add lits.* **(Refer section 10.2)**
12. *Consider the following representation on from blocks world.*

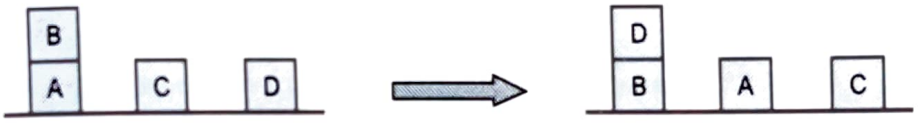
    Start : $ON(B, D) \wedge ON(C, B) \wedge ONTABLE(D) \wedge ONTABLE(A)$.

    Goal : $ON(A, B) \wedge ON(C, D) \wedge ONTABLE(B) \wedge ONTABLE(D)$.

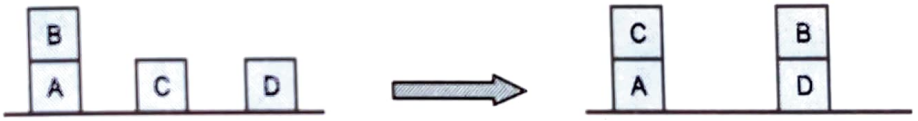
    i) Show how STRIPS would solve this problem.

    ii) Did these processes produce optimum plans and if not justify how it can be done ?

    (Refer section 10.2)
13. *Explain goal stack planning. Solve the following using goal stack planning.* **(Refer section 10.2)**



14. *Solve the following using goal stack planning.* **(Refer section 10.2)**



15. *Consider the problem of swapping the contents of two registers A and B. Suppose that the single operator ASSIGN (x, v, iv, 0v) is available which assigns the values v, which is stored in location 1v to location 2 which previously contained the value 0v.*

    ASSIGN (x, v, 1v, 0v)

    P : CONTAINS (1v, v) ^ CONTAINS (x, 0v)

    D : CONTAINS (x, 0v)

    P : CONTAINS (x, 1v)

    Assume that there is atleast one additional register C available.

    a) What should STRIPS do with this problem 2 ?

    b) How might you design a program to solve this problem ? **(Refer section 10.2)**
16. *Write a note on reactive system.* **(Refer section 10.6)**

10.8 University Questions with Answers

Summer - 14

Q.1 *Explain goal stack planning using suitable example.* **(Refer section 10.2)** **[7]**

Summer - 16

Q.2 *Explain steps of natural language processing.* **(Refer section 10.1)** [7]

Q.3 *Write a short note on : Hopfield networks.* **(Refer section 10.1)** [7]

Summer - 18

Q.4 *Discuss nonlinear planning using constraint posting with exampl.* **(Refer section 10.4)** [7]

Winter - 18

Q.5 *Discuss goal stack planning.* **(Refer section 10.3)** [4]

❑❑❑