Chapter - 13 of book & 10th of syllabus 89

Saathi

Date __/__/____

Planning

→ To solve any nontrivial problem, it is necessary to combine some of the basic problem solving technique & one of the knowledge repre^{n} mechanism.

→ It's also useful to divide problem into smaller pieces & then solve these pieces separately to the possible limit.

13.1 ☆ Overview

→ We can describe a process of problem-solving as a search through a state-space in which each point corresponds to a state. The search started with an initial situation & performed a sequence of opera^{n}s until a goal state was reached.

→ For ex., the A* algo. provides a way of conducting Best-First Search through a graph representing a problem space.

→ However, for more complicated problem domains, it becomes imp. to be able to work on small pieces of problem separately and then combining partial solu^{n}s can lead to complete problem solu^{n}.

→ First of all, we must avoid to recompute entire problem state, when we move from one state to the next. Instead of this, we should consider only part of state that may have changed. For ex, if we move form one room to another, this doesn't affect loca^{n}s of doors & windows in the rooms.

→ Several methods for doing decomposition have been proposed. These methods focus on ways of decomposing original prob. into appropriate subproblems. It also focuses on handling interaction among subparts. The use of these methods is often called "planning."

→ In regular use, the "planning" refers to the process of computing several ~~steps~~ steps of procedure before executing any of them. For ex., in solving 8-puzzle, it can't actually push any tiles around. So, when we talk about computer soln of 8-puzzle, then we generate a plan for solving it. For problems such as 8-puzzle, the difference bet" planning & doing is not imp., bcoz in this problem, the steps can be undone.

→ If soln steps in real world can't be ignored or undone, then planning becomes extremely imp.

→ In next topics, a variety of planning techniques are presented. All of them, except the last one, are prob.-solving methods that rely heavily on prob-decomposition.

13.2 An example domain : The blocks world

→ In blocks-world problem, there is a flat surface on which blocks can be placed. There are no. of square blocks of same size. They can be stacked one upon another.

→ There's a "robot arm" (Robot ka ek hath) that can manipulate the blocks. The foll. actions can be performed:-

Date __/__/____

Saathi

- UNSTACK (A,B) :- Pick up block A from its current pos^n on block B. The arm must be empty & block A must have no block on top of it. condition
- STACK (A, B) :- Place block A on block B. The arm must be holding & surface of B must be clear. condition
- PICKUP(A) :- Pick up block A from the table & hold it. The arm must be empty & there must be nothing on top of A. condition
- PUTDOWN(A) :- Put block A down on the table. The arms must have been holding block A. condition

→ Notice that, in fact, the robot arm can hold only one block at a time.

→ To specify conditions under which opera^n may be performed and results of performing it, we need to use foll. predicates.

- ON(A,B) — Block A is on block B.
- ONTABLE(A) — Block A is on the table.
- CLEAR(A) — There's nothing on top of block A.
- HOLDING(A) — The arm is holding block A.
- ARMEMPTY — The arm is " nothing.

→ Various logical statements are also true in blocks world. For ex.,

$[\exists x : HOLDING(x)] \rightarrow \neg ARMEMPTY$:- Arm is holding anything, then it's not empty.

$\forall x : ONTABLE(x) \rightarrow \neg \exists y : ON(x,y)$:- If block is on table, then it is not on another block.

$\forall x : [\neg \exists y : ON(y,x)] \rightarrow CLEAR(x)$:- Any block (with no block on it) is clear.

Page No.

Saathi

Date __/__/____

B.3

☆ Components of a planning system

→ In problem-solving systems, it's necessary to perform each of the foll. func^{n}s.

(1) Choose the best rule to apply next, based on the best available heuristic func^{n}.

(2) Apply the chosen rule to compute the new prob. state that arises from its app.

(3) Detect when a solu^{n} has been found.

(4) Detect dead-ends so they can be abandoned.

→ In more complex system, the 5th opera^{n} is also imp.

(5) Detect when an almost correct solu^{n} has been found & employ special technique to make it totally correct.
Use

(1) Choosing rules to apply :-

→ The most widely used technique for selecting appropriate rules to apply is bet^{n} desired goal state & current state.

→ If several rules are found, then a variety of heuristic info. can be used to choose among them.

→ For ex., if our goal is to have a white fence around yard & we have brown fence, then we select operator to involve change of colors of an obj. If we don't have fence, then we consider operators to construct fence first.

Page No.

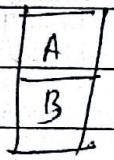
Saathi

Date __/__/____

(2) Applying rules :-

→ In simple systems, applying rules was easy. Each rule simply specified the prob. state that would result from its app^{n}s.

→ Fig. shows the "state" s of simple blocks-world problem.



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

→ The effect of the operator UNSTACK(X, Y) could be described by foll. axiom.

$$[CLEAR(X, S) \wedge ON(X, Y, S)] \rightarrow$$
$$[HOLDING(X, DO(UNSTACK(X, Y), S)) \wedge$$
$$CLEAR(Y, DO(UNSTACK(X, Y), S))]$$

→ Here, DO is a func^n that specifies for given state & given action. The axiom states that if CLEAR(X) & ON(X, Y) both hold in state "s", then HOLDING(X) & CLEAR(Y) will hold in the state from DOing an UNSTACK(X, Y) operation in state 's'. (જો 'x' clear હોવા માટે તો 'x' જો 'y' ની ઉપર હોવા, તો UNSTACK(X, Y) operation કરવાથી 'x' hold થવા માટે 'y' ની ઉપર કંઈ ના હોવા, એટલે તો clear હોવો જોવા).

Date __/__/__

(3) Detecting a soln

→ A planning system has succeeded in finding a soln to a problem when it has found a sequence of operators that transform initial problem state into goal state.

→ How will system know when this has been done? In simple problem-solving systems, this question can be answered easily by a straightforward match of states; but not for complex systems.

→ One representational technique has served many planning systems. Suppose that, we have P(x) as predicate as part of goal. To see whether P(x) is satisfied, then we ask whether we can prove 'P(x)' by given assertions or not.

→ If we can construct such a proof, then problem-solving process terminates. If we can't then a sequence of operators which can solve prob. must be proposed.

(4) Detecting dead ends

→ Since planning system is a searching for sequence of operators to solve a particular problem, it must be able to detect paths that can never lead to a soln.

→ If search process is reasoning forward from initial state, then it can prune path that leads to a state from

which the goal state can't be reached. For ex., suppose we have fixed supply of paint: some white, some pink & some red. We want to paint room with light red walls & a white ceiling. We can produce light red by mixing some white paint in red. But we can't paint ceiling by mixing "pink" & "red" colors.
'By combining 'pink' & 'red' colors, we will never get solutions,' so prune that path.

→ If we reason backward from goal state, then it can terminate path either bcoz it's sure that the initial state can't be reached. For ex., in trying to satisfy goal A, the program eventually reduces its problem to the satisfaction of goal A as well as goals B and C, since it has made a little progress. It has produced problem even harder than original one, so this path should be avoided.

(5) Repairing an almost correct soln

→ Two kinds of techniques are often useful in solving nearly decomposible problems.

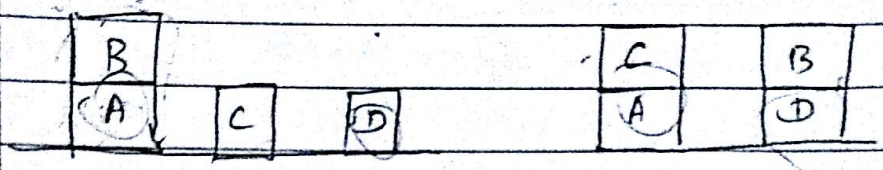
One way is to assume that probs are completely decomposible, proceed to solve subproblems and then, sub-soln's are combined to yield soln of orginal prob.

A better approach is to look & observe the situation that results when sequence of operations is executed & compare this situation with desired goal. In most cases, this diff will be smaller than diff. bet" initial & goal state.

Saathi

Date __/__/____

13.4 ☆ Goal stack planning



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

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

→ When we begin solving this prob, the goal stack is simply
ON(C,A) ∧ ON(B,D) ∧ ONTABLE(A) ∧ ONTABLE(D)

→ But we want to separate this prob. into four subprob., one for each component of the original goal.

→ Two of the subprobs, ONTABLE(A) & ONTABLE(D) are already 'true' in the initial state. So, we will work only on remaining two.

→ Depending on the order in which we want to tackle the subproblems, there are two goal stacks that could be created as our 1st step, where each line represents one goal on the stack & GTAD is an abbreviation for ONTABLE(A) ∧ ONTABLE(D).

A-D = ON table & A clear

ON(C,A)	ON(B,D)
ON(B,D)	ON(C,A)
ON(C,A) ∧ ON(B,D) ∧ GTAD	ON(C,A) ∧ ON(B,D) ∧ GTAD
①	②

Saathi

Date __/__/____

→ Let's first assume that alternative-① . Alternative-② will also lead to a solu^n.

→ Let's first explore alternative-①. We first check to see whether ON(C,A) is true in current state; but it is not true.

→ Out of the operators, let's call one another operator STACK; which has to be called by C & A. So, we place STACK(C,A) on the stack in place of ON(C,A), so we obtain...

STACK(C,A)
ON(B,D)
ON(C,A) ∧ ON(B,D) ∧ OTAD

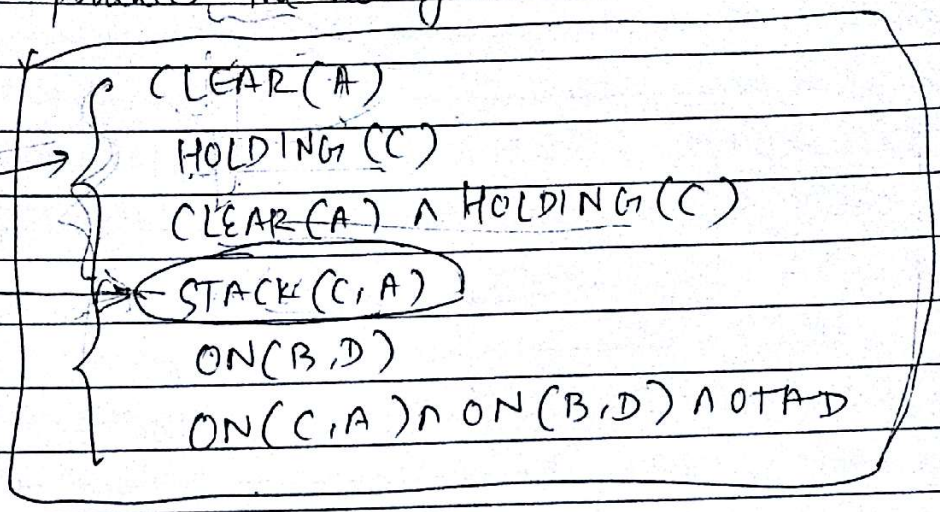
→ STACK(C,A) is replaced by ON(C,A), bcoz after performing the STACK(C,A), we are guaranteed that ON(C,A) will hold. So, to apply STACK(C,A), then foll. precondition must be held.

CLEAR(A) ∧ HOLDING(C)

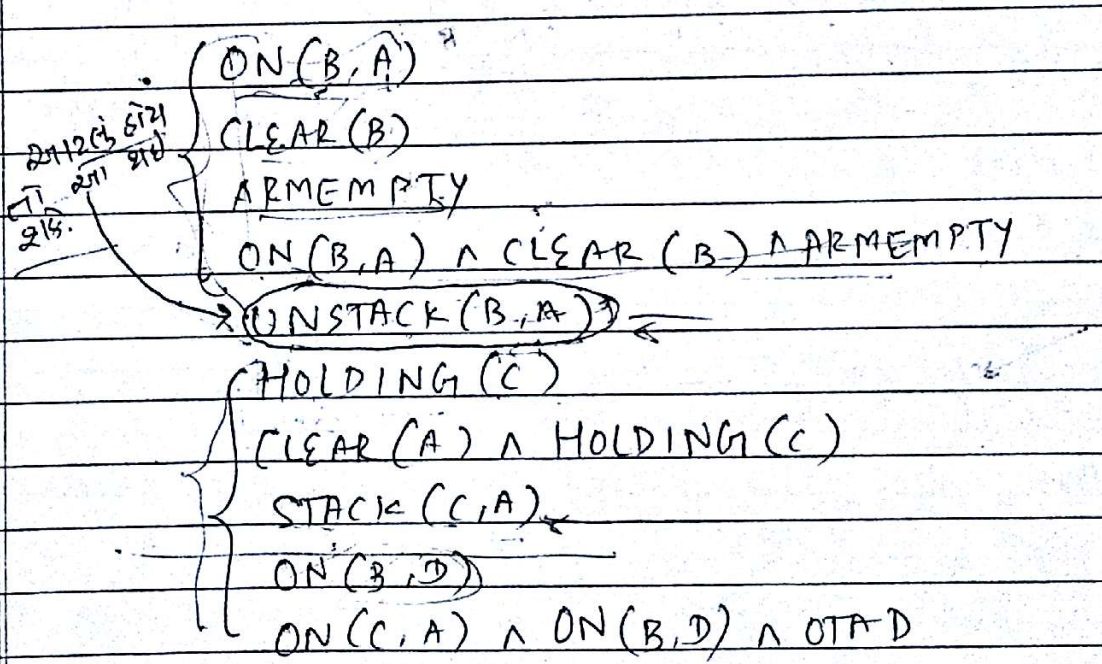
→ At this point it's useful to exploit some heuristic knowledge. HOLDING(x) is very easy to achieve. At most, it is necessary to put down something else & then to pick up desired object. But, HOLDING is also very easy to undo. To anything else, robot will need to use arm. So, if we achieve HOLDING first, then try to do something else, then HOLDING won't be true anymore. Therefore, HOLDING must be tackled last.

Date ___/___/_____

→ This produces the new goal stack:-



→ Next we check to see if CLEAR(A) is true. But it's not. The only operator that could make it true is UNSTACK(B,A). So we will attempt to apply it. This produces "goal stack" as follows:-



→ This time, when we compare top element of goal stack, ON(B,A) to initial state, we see that it's satisfied.

→ So, we pop off ON(B,A) & considers next goal CLEAR(B). It's also satisfied, so popped off.

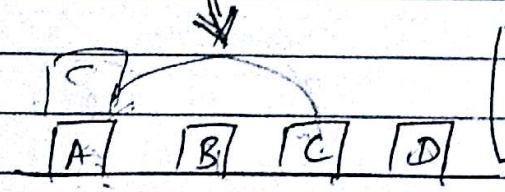
Date ___/___/_____

Saathi

→ The third is ARMEMPTY, which is also a 'pre-condition' for UNSTACK(B,A). It's also true in the current world model, so it can be popped off the stack.

→ The 4th statement is combined goal representing all of the preconditions for UNSTACK(B,A). We make sure d it is satisfied in initial state.

→ Now ~~if~~ the top element of stack is operator UNSTACK(B,A). We are now guaranteed that "pre-cond's" are satisfied. So, it can be applied to produce a new "state" from which rest of the prob. solving process can continue.



After applying UNSTACK(B,A) operator a holding 'B' block

→ ~~Likewise~~ ~~Still~~ We have popped off the stack up to ON(B,D). But ON(B,D) condition still does not hold. So, we can solve this sub-goal "ON(B,D)" as the previous sub-goals i.e "ON(C,A)".

Saathi

Date __/__/____

13.5 ★ Non-linear planning using constraint posting

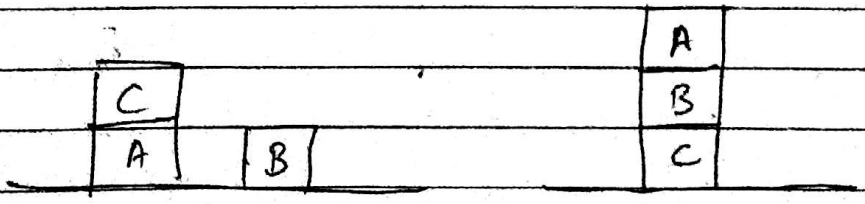
→ The goal-stack planning method attacks problems by solving the goals one at a time in order.

→ A plan generated by this method contains a sequence of operators for attaining first goal, and then followed by a sequence of second goal etc...

→ The operators used to solve one subprob may interfere with the soln of second subprob.

→ If we want to work with a plan in which multiple subproblems are worked on simultaneously. Such a plan is called a "non-linear plan", broz it is not composed of linear sequence of subplans.

→ Let's incrementally generate a non-linear plan to solve the foll. problem.



Start: ON(C,A) ∧
ONTABLE(A) ∧
ONTABLE(B) ∧
ARMEMPTY

goal: ON(A,B) ∧
ON(B,C)

→ We begin with the null plan, i.e. a plan with no steps. Next we look at goal state & steps for achieving that goal.

Page No.

→ We will solve two subproblems of goal state with respective post-cond^n's ON(A,B) & ON(B,C).

	Sub-problem-1	Sub-problem-2
Pre-cond^n	CLEAR(B) * HOLDING(A)	CLEAR(C) * HOLDING(B)
	STACK(A,B)	STACK(B,C)
Post-cond^n	ARMEMPTY ON(A,B) ¬ CLEAR(B) ¬ HOLDING(A)	ARMEMPTY ON(B,C) ¬ CLEAR(C) ¬ HOLDING(B)

→ Each step is written with its pre-cond^n above it & post-cond^n below it.

→ Notice that, the steps are not ordered with respect to each other. All we know is that we want to execute both of them eventually.

exe^n order doesn't matter

→ An unachieved pre-cond^n is marked with star (*). Both of the *HOLDING precond^{n}s are not-achieved, bcoz the arm holds nothing in the initial problem-state.

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 in the appropriate details.
- Early attempts to do this involved the use of macro operators, in which larger operators were built from smaller ones.
- In this approach, no details were eliminated from actual descriptions of the operators.

ABSTRIPS

- A better approach was developed in ABSTRIPS systems which 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.
    - Augment the plan with operators that satisfy those preconditions.
    - 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.
- The assignment of appropriate criticality value is crucial 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.

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 inside a room.
- Reactive systems are capable of surprisingly complex behaviors.
- 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.

Other Planning Techniques

- Triangle tables
    - Provides a way of recording the goals that each operator is expected to satisfy as well as the goals that must be true for it to execute correctly.
- Meta-planning
    - A technique for reasoning not just about the problem being solved but also about the planning process itself.
- Macro-operators
    - Allow a planner to build new operators that represent commonly used sequences of operators.
- Case based planning:
    - Re-uses old plans to make new ones.