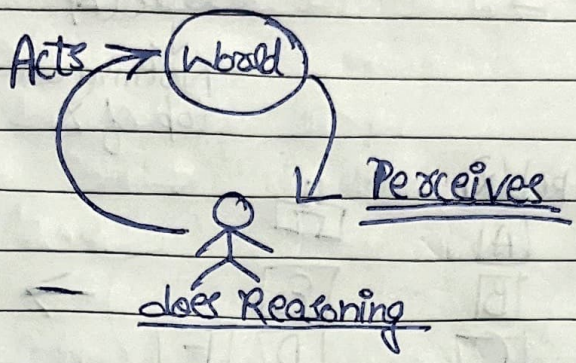
Page No.
Date

Planning

⇒ The reasoning side of acting is planning.



In planning problem, we will have

⇒ Domain Description
Action Specification
Goal states description
↔ Planning Domain Description language (PDDL)

⇒ Planner will be domain independent.

PDDL 1.0 ⟹ STRIPS domain Planners (for describing planning domain)

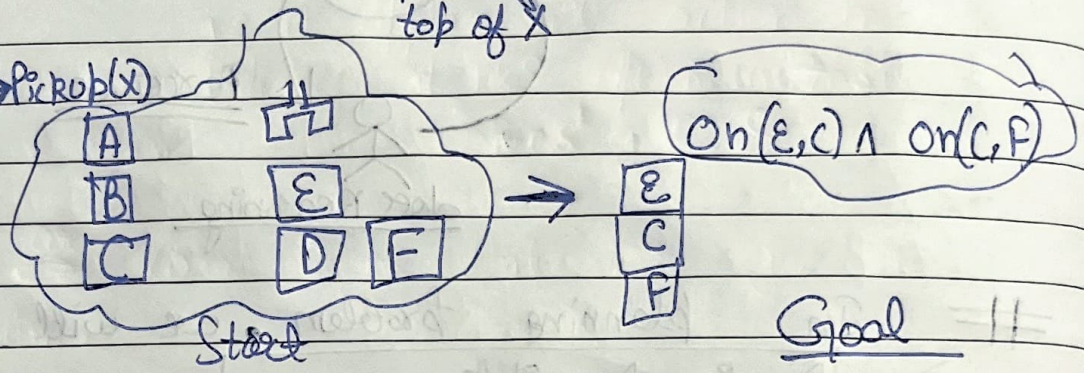
Assumptions:-

1. Space is finite
2. Fully observable. → Agent can sense the entire world.
3. Static
   ↓ Agent is only one making changes in the world.
4. Deterministic → Whatever actions agent does they will happen in real world.
5. Simple goals.
6. Plan → Sequence of actions
7. Instantaneous actions. (No time)

⇒ Let's see how Box world is described in Strips :-

⇒ Domain Predicates → On(X,Y) (X on top of Y), OnTable(X) (X on table), Holding(X) (Robot arm holding X), ArmEmpty (AE), Clear(X) (Nothing on top of X)

⇒ Actions → PickUp(X)



⇒ Using predicates we can describe the states like On(A,B), On(B,D), onTable(C) ... Clear(A) etc.

→ Actions → For describing actions :-
↳ What is true for action to be applicable ?
↳ What will become true after action is applied.

In STRIPS → For Action ⇒ ① PickUp(X)

Lists :
① Precondition ⇒ OnTable(X), Clear(X), AE
② Add : Holding(X)
③ Delete : OnTable(X), AE

Here, F could be picked

∵ Preconditions satisfied

② PutDown(X)
Precondition : Holding(X)
Add : OnTable(X), AE
Delete : Holding(X)

...roses ⇒ likes(steve, easycour...
∧ ...(Science Courses)

Page No
Date

(3) Unstack (X, Y) : Unstack x from Y
(4) Unstack (Y, X) : Stack x on Y

In modern for action (a) we use,

① pre(a)
② effects$^+$(a) } effects
③ effects$^-$(a) }

Planning problem → Domain description → D
operators → O
Start → S
Goal → G

~~→ Described by (O, D, S, H)~~
→ Described by (O, D, S, G)

S
& G is set of predicates

⇨ Action (a) is applicable if Precond(a) ⊆ S
When a is applied we Progress to S'

$$S' \leftarrow (S \setminus \text{effects}^-(a)) \cup \text{effects}^+(a)$$

A plan $\pi = (a_1, a_2 \ldots a_n)$ [seq. of actions]

#GoalTest (G) ⊆ S then S is Goal state.
↓
Set of pred. describing the goal

Page No.
Date

Having a Goal test fⁿ & somebody gives me a plan π

⇒ How to validate that plan π is correct i.e it solves the problem?

↳ Write a small program that will progress over the state

i.e it applies all action in seq. & sees if the result is correct

i.e $g \subseteq S$

Now, we want algos that will give us the plan.

Stack(X, Y)

Pre :- Holding(X), clear(Y)

effects⁺ : On(X, Y), AE, clear(X)

effects⁻ : Holding(X), clear(Y)

UnStack(X, Y)

Pre :- On(X, Y), AE, clear(X)

effects⁺ : Holding(X), clear(Y)

effects⁻ : On(X, Y), AE, clear(X)

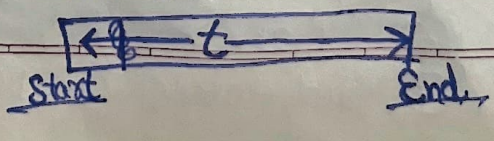
Richer Domains Actions :-

⇒ Conditional effects →

effects which come into force only if certain cond^n are true. (if)

⇒ Durative actions (Time taken into account)

represented by intervals



Here, we have Start Preconditions + End Precondⁿ + Effects + Effects + overall condⁿ

⇒ Metric Values → Keeps track of values throughout the duration

These are not considered in our case

We will focus on STRIPS PDDL domain only

Action a is applicable if pre(a) is a ⊆ S in state S.

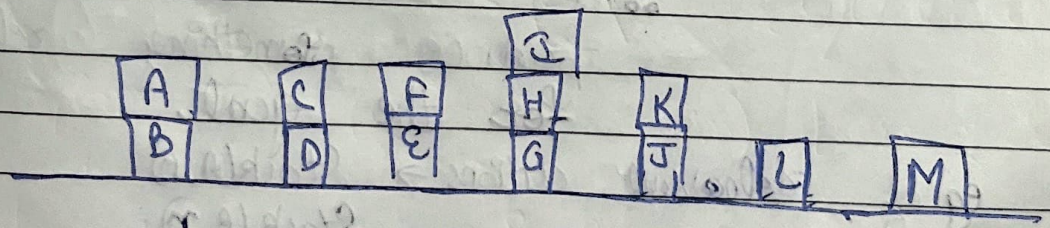
→ progress (S, a) $S' \leftarrow (S - effect^{-}(a)) \cup effects^{+}(a)$

~~Goal g~~

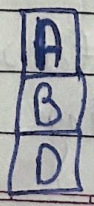
→ State S satisfies a goal g if $g \subseteq S$

Planning problem → (Start S, goal g, actions)

Start State :-



Goal State →



FSSP does

⇒ Forward State Space Search :-

↳ MoveGen in Rose would generate all the possible actions :-
Unstack A_1, C_1, F_1, I_1, K
or Pickup L, M

Say, K is Pickedup & S updated to S'
again moveGen on S' & so on

Problem → Large branching factor
∴ brute force ✗
① ∴ do Heuristic Search (Not in Syllabus)
(We want domain independent heuristics)
② do Backward Search

BSSP does

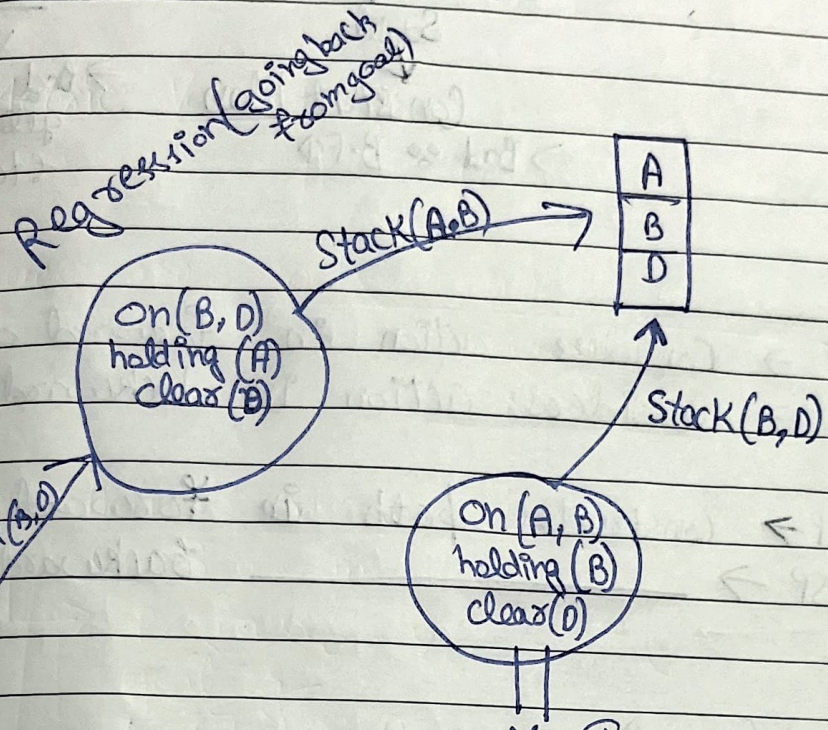
⇒ Backward State Space Search

Relevant actions →
a is relevant for goal g iff
$effect^+(a) \cap g \neq \phi$
& $effect^-(a) \cap g = \phi$
∴ we want those actions that produce something relevant for the goal.
eg of relevant actions → Stack(A, B)
Stack(B, D)
∴ here BSS is focused on goal
∴ small branching factor

→ not sound (What we get is not state)

regress (g, a)

$g' \leftarrow (g - effects(a)) \cup pre(a)$



Disadvantage → what we get are not state essentially.

⇒ terminates when $g''' \subseteq$ Start

Check for Valid Plan :-

π, (S, g, action)

Begin with Start state

Keep progressing with actions & see the state reached in goal or not

$a_1, a_2, a_3 \cdots a_k$

Check for Consistency & Prune States :-

Deletes the inconsistent node path.

Page No.
Date

↳ ∴ We want mix of FSS & BSS

FSS → Sound → Consistent plan ⇒ Bad ∵ B.F ↑

BSS → Low B.F → ⇒ Bad ∵ we get wrong state format.

↳ In FSSP → Considers action in forward dir^n
BSSP → Considers action in backward dir^n

↳ In FSSP → Constructs path in forward dir^n
In BSSP → _____ backward dir^n.

↳ In BSSP (regress)
↳ relevant action ⇒

$effects^+(a) \cap g \neq \phi$
$effects^-(a) \cap g = \phi$

↳ Regression

regress (g, a)
$g' \leftarrow [g - effects^-(a)] \cup precond(a)$

In FSSP (progress)

$s' \leftarrow (s - effects^-(a)) \cup effects^+(a)$

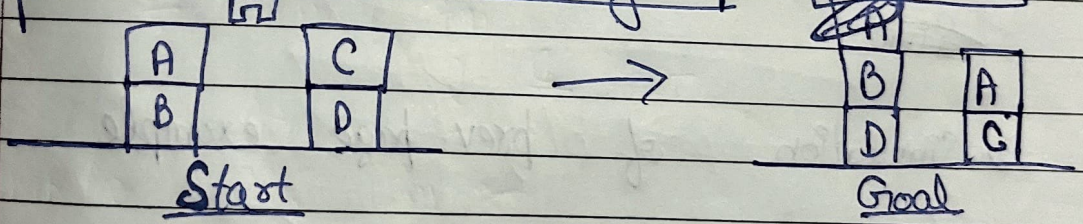
⇒ In Construction of plan in forward dirⁿ, we will always end up with states. ✓

⇒ But, in backward SSP we start constructing from end & what we get might not be a state, hence we do validity check.
∴ (X)

⇒ In FSSP, by considering actions in forward dirⁿ ⇒ B.F ↑
But, in BSSP by considering actions in backward dirⁿ B.F ↓

∴ We want construction in forward dirⁿ (S→G) & actions in backward dirⁿ, (Goal oriented)

Goal Stack Planning :- OR LINEAR PLANNING



goal ≡ on(A,B)
∧ on(B,D)
∧ clear(A)

Algo →
Push goal(s) onto Stack
Pop Stack
If predicate :
If true (Cond^{n} is already true)
do nothing
else
Push relevant action onto stack

Page No.
Date

Else (not predicate means action)

Algo →

Push goals onto Stack
Pop Stack → Push each predicate
If predicate
If true (predicate cond^n already true)
do nothing
else
Push relevant action onto stack
Push precond(a) onto Stack → Push each predicate of precond(a)
Serializing the goals
Else (Action)
Add action to plan
$\pi \leftarrow \pi \cdot a$ (· → concatenate)

⇒ Simulation of prev page example

Initial Sub Goals →
- On AB ∧ on BD
- on BD
- on AB

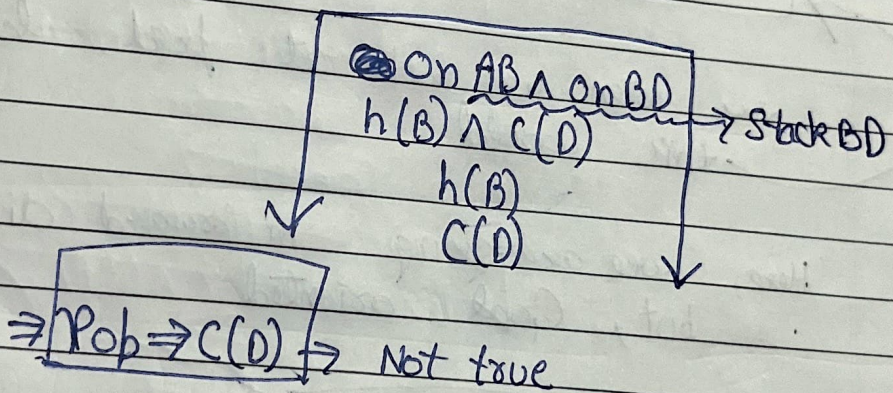
on AB popped out
↳ predicate
& true in given ∴ do nothing

On BD → predicate not true for given state
∴ we push an action which will make this true

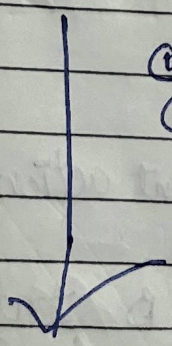
OnBD → action → Stack BD
↳ pushed

Precond → holding (B) [h(B)]
∧ clear (D) [C(D)]
Also h(B)
C(D)

∴ Current Stack →



∴ push an action to make this true.



① Unstack (CD)
② Precond → On CD ∧ AE ∧ clear C
AE
On CD
C(C)

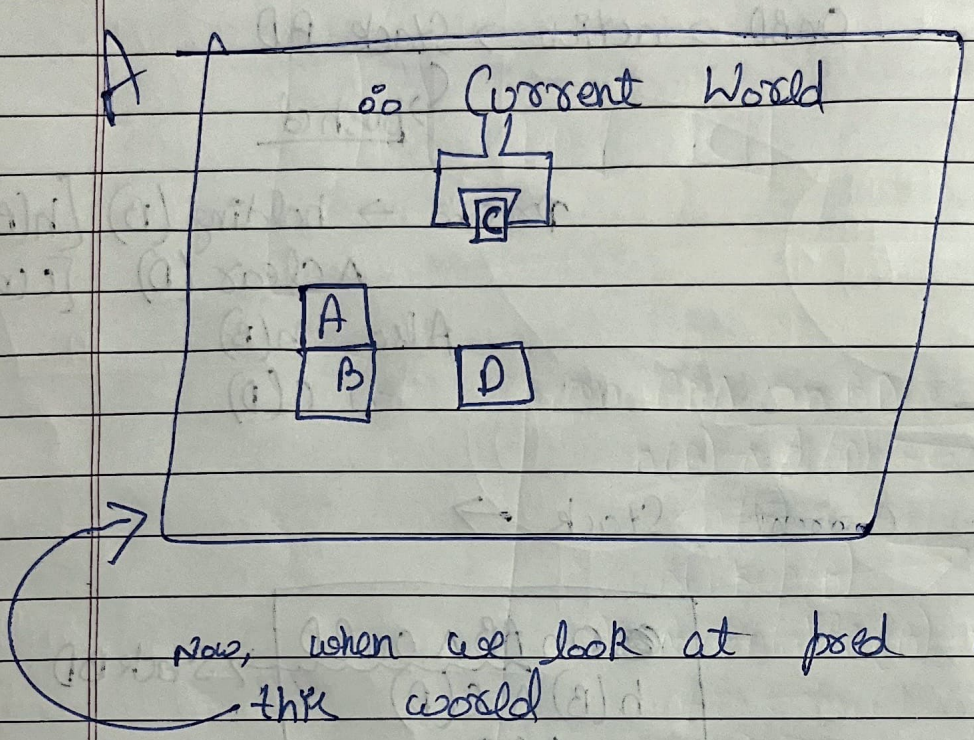
Pop C(C) → True → Pred
Pop On(D → True → Pred
Pop AE → True

Page No.
Date

Pop $On(C,D) \wedge AE \wedge clear(C) \rightarrow$ ~~Action~~ Pred $\Rightarrow$ True

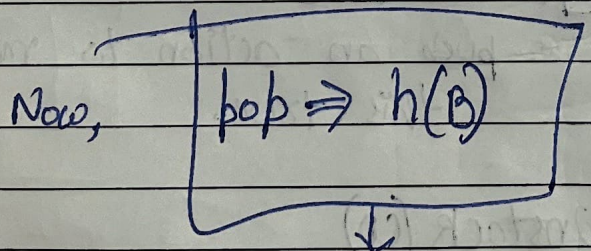
Pop Unstack (C,D) → Action (1)

∴ Add this to plan



Now, when we look at pred look at this world

Here, we are going in forward dirⁿ but, Goal oriented



Not true

∴ Insert relevant action

Here we have a choice

To make holding B true make an action Unstack (X, B) or Pickup (B)

Say we choose Pickup (B) [By looking at State]

Pickup (B)
AE ∧ Ontable(B) ∧ clear(B)
AE
on(B)
C(B)

Pop C(B) → Not true
∴ Another action

Unstack (A,B)
on(A,B) ∧ AE ∧ clear(A)
AE
on(A,B)
clear(A)

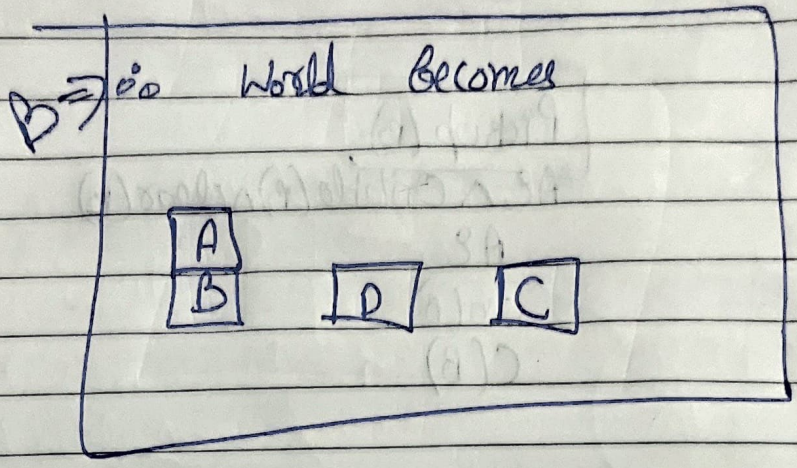
clear(A) → true → popped out
on(A,B)

AE → not true.
Another action

PutDown (c) → (2)
h(c)

Pop h(c) → true
Do action PutDown(c)

~~Stack Current~~



Stack ⇒

On AB ∧ On BD → Stack BD

~~h(B) ∧ c(D)~~

~~h(B)~~

~~Pickup(B)~~

~~AE ∧ Ontable(B) ∧ clear(B)~~

~~AE~~

~~On(B)~~

~~Unstack (A, B)~~

~~On(AB) ∧ AE ∧ C(A)~~

~~AE~~

~~On(AB)~~

~~C(A)~~

~~Putdown (C)~~

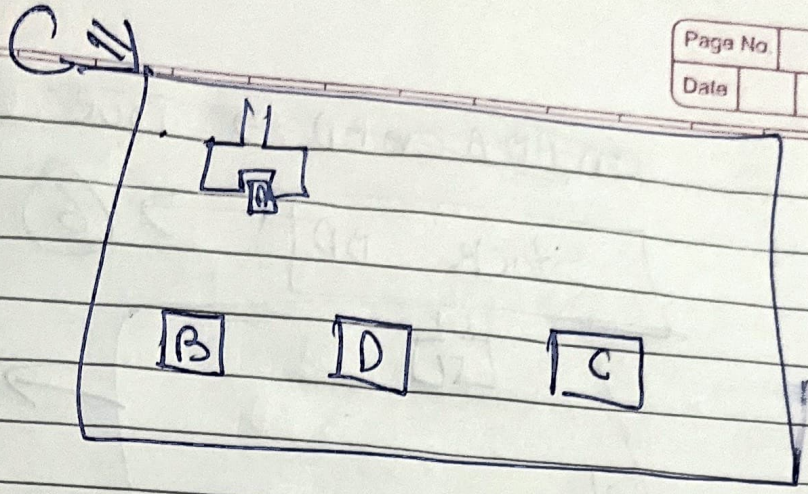
~~h(C)~~

Pop On(AB) ∧ AE ∧ C(A) → True

Pop Unstack (A, B)

↳ Action

3

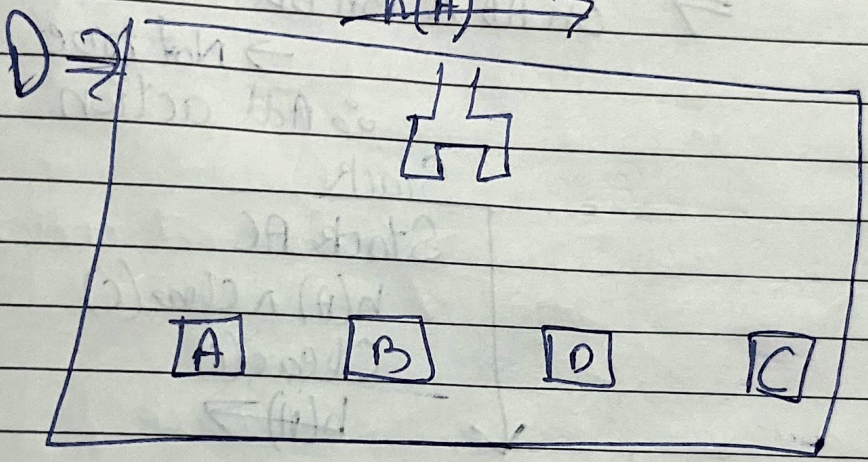


On(B) → True

AE → Not true

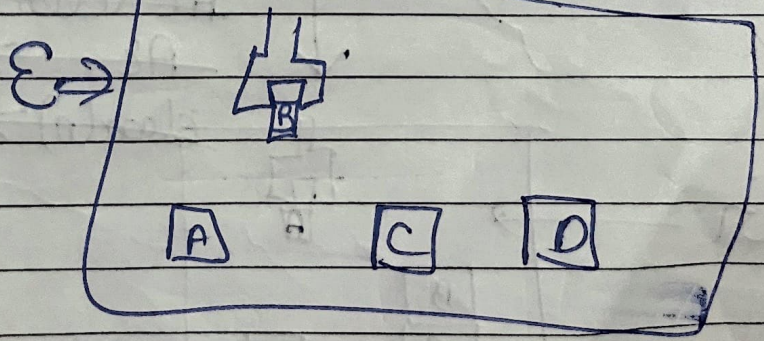
~~Putdown(A)~~ → executed

~~h(A)~~ →



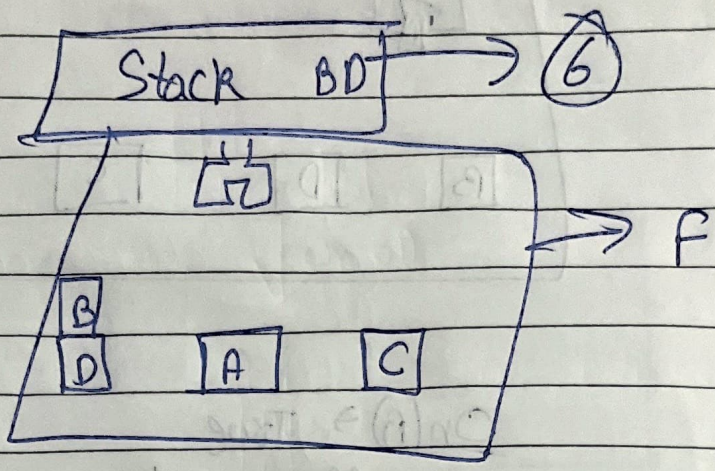
AE ∧ Ontable(B) ∧ Clear(B) → True

5 ← Pickup(B) → Action



h(B) ∧ c(0) → True

~~On AB ∧ On BD → True~~



⇒ On AB ∧ On BD

→ Not true

∴ Add action

Stack

Stack AC

h(A) ∧ Clear(C)

Clear(C)

~~h(A)~~ →

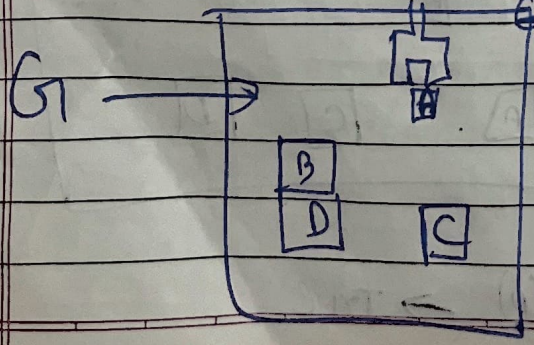
h(A) → Not true

~~Pickup(A)~~ 7

~~Clear(A)~~ → True

~~AE~~ → True

~~Clear(A)~~ → True



... (already covered)

Page No.
Date

H

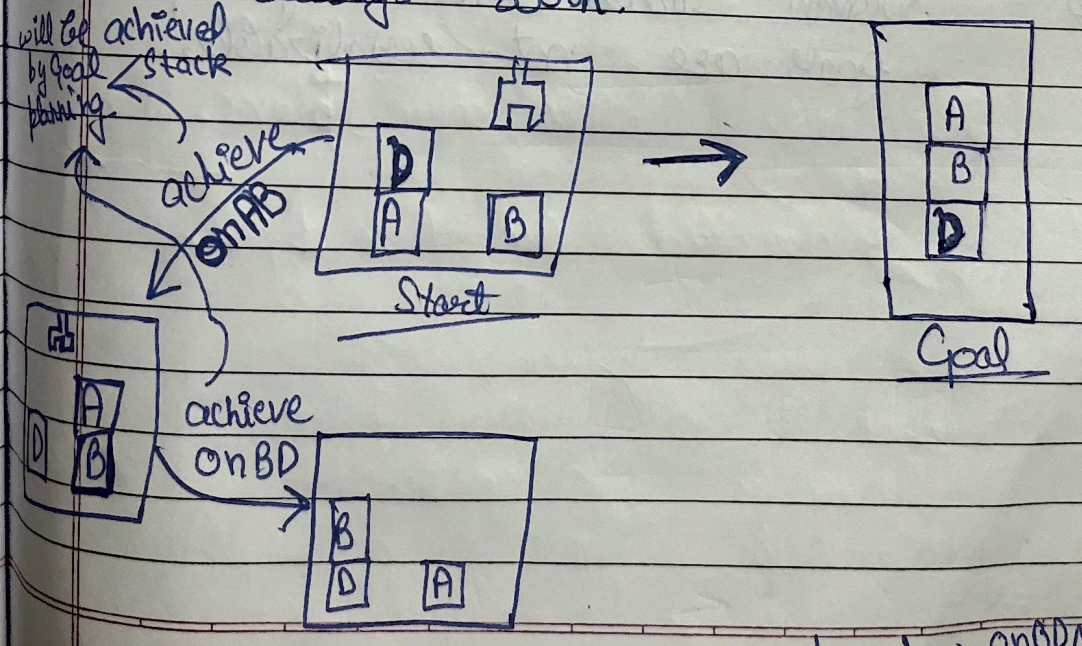
clear(C) ✓
h(A) ∧ clear(C) ✓

(8) → Stack(AC) → Executed
stack empty
terminate

∴ Plans considered in backward fashion (construction of Plan) ✓ But action taken in forward

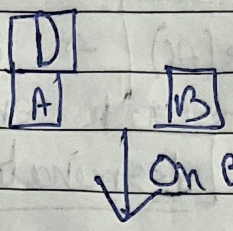
⇒ Ordering how to perform action is important here.
⇒ Here, we are ~~sorting~~ serializing the Goals, hence known as (Linear planning)
⇒ SUSSMAN's Anomaly

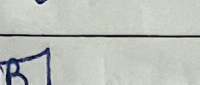
↳ It shows that Goal stack will not always work.

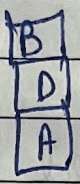


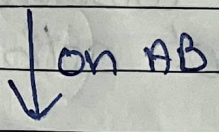
∴ We achieved ~~how~~ onBD ∧ onAB but Not Goal State

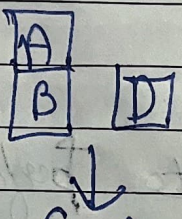
∴ This order not correct
let's try other order











Goal not achieved

We can do extra work but Algo wrong here

⇒ Sussman showed in many problems goals are not serializable.

Non LINEAR PLANNIG (Partial Order Planning) (NLP)

↳ Least Commitment Planning
(Very less commitments)
(We will add only limited stuff to plan)

↳ Plan Space Planning

↳ Partially specified ⇒ Refinement step Plan

⇒ Partial Plan is denoted as a 4 tuple

$<A, \alpha, C, B>$ (all are set)

↳ A → Actions part of plan

↳ α → Ordering relation which specifies order of actions.

↳ C → Causal links

↳ B → Binding Constraints

⇒ Say Current action =
Stack(A, x)
then we can put constraints like
$x = B$
$x \neq B$
$x \in \{B, C\}$

Causal Link is a triple where
$<a, P, b>$

<a, p, b> means
a produces predicate p
& b consumes predicate p

a, b are actions

example →
<Unstack(A,y), holding(A), PutDown(A)>

⇒ p ∈ effects(a)
& p ∈ precond(b)

⇒ In Goalstack planning we commit to solve one subgoal first and then the second subgoal.
But in partial plan limited commitments.

⇒ we are not working in space of states anymore, we will start with some initial plan and we will apply refinement step to get new plans.
Search space contains only plans. (Partial plans)
↳ described by four tuple.

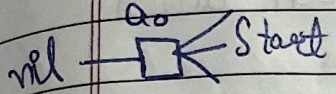
What we want to do:
↳ look at a partial plan
↳ Choose a refinement step
↳ Come up with new partial plan

⇛ When to terminate :-

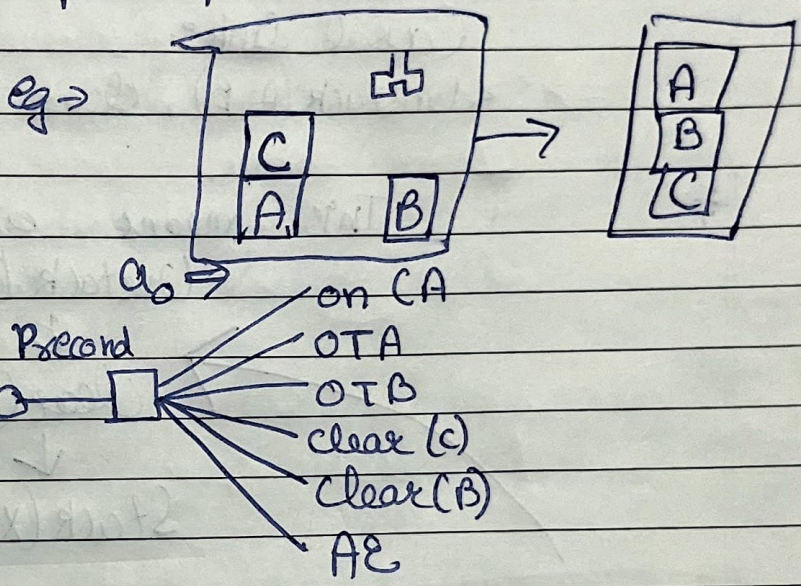
↳ Initial plan – Π_0 has two actions & one ordering constraint.

$$\langle (a_0, a_\infty), a_0 < a_\infty, (), () \rangle$$

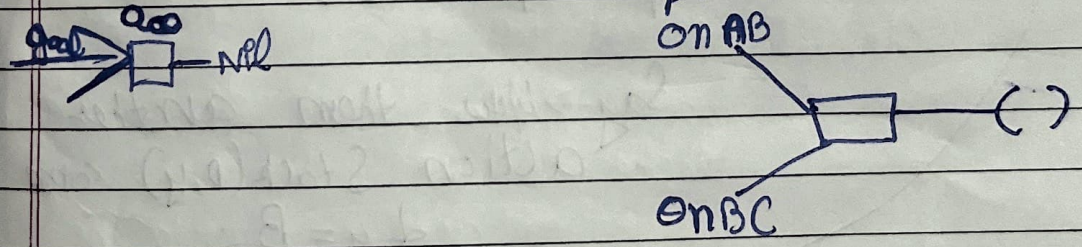
↳ a_0 → no preconditions
effects = {start}



It simple produces start state



↳ a_∞ → no effects
precond →



↳ Π_0 can be seen to stand for set of plans in which a_0 is first action & a_∞ is last

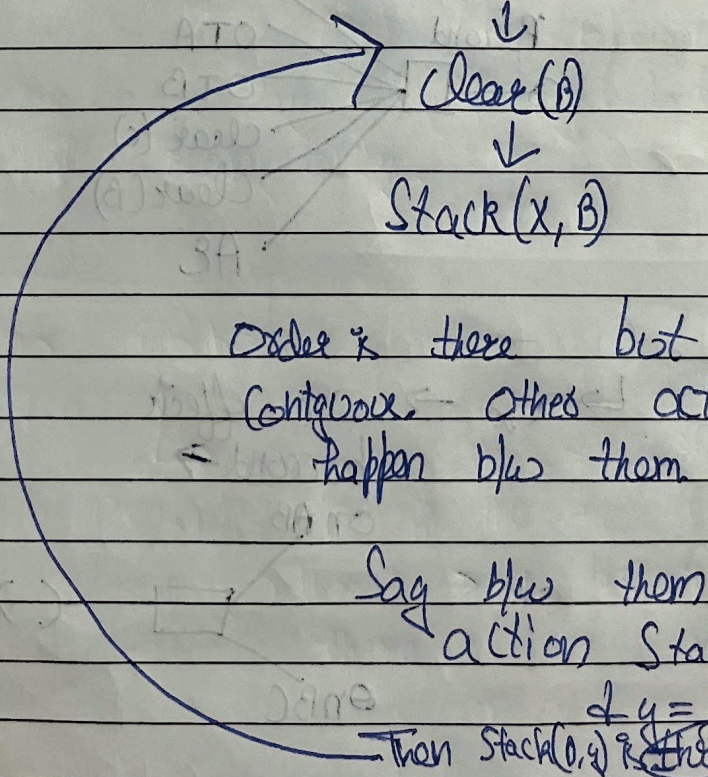
Now, the goal is ~~to~~ make connections b/w ao tap

↳ We want to add Causal links such that they are not disturbed later on.

Problem:
Say we want for an action

Causal link:
<Unstack(A,B), clear B, Stack(x, B)>

This means ordering



Order is there but not contiguous. Other actions could happen b/w them.

Say b/w them another action Stack(O,y) come & y = B
Then Stack(O,y) is threat.
∴ Causal link will be disrupted by this action.

⇒ Planning process should consider this ~~into action~~ & do something about it.

⇒ TWEAK was one of the first NCPlanning algo.
↳ There they use the term CloBBering/DECLOBBERING which modern algos don't use.

⇒ CLOBBERING means I am CLOBBERING the Causal link. If it CLOBBERS it then another action will DECLOBBER it. (Causal link)
Hence Causal links are not disrupted.

⇒ Ordering links should be consistent. (No cycle)

Refinement steps are of 4 types
↳ Add action to A
↳ Add ordering link to A
↳ Add Causal link to C
↳ Add a Constraint
⇒ We could choose any one at a time.

Imp
⇒ We will start with initial partial plan π_0
then we start to fill in
↳ How to fill in?
↳ We have series of refinement Steps.

When to terminate? Solution
↳ State Space perspective
↓
~~Simple~~ Complete B
↓
Topological Sort of $\pi \rightarrow \pi'$
↓
Progress from start State & apply action.

Page No.
Date

Many topological sort possible so ignore.

Other way:

Solution ?

↓

Has No flaws

⇒ We want to look at partial plan & want to say whether solution or not

⇒ Implicitly we are doing the prev. page thing only.

i.e, if this were a set then any consistent ordering of action will be valid.

But, here we say this by looking at partial plan only.

Has no flaws :-

→ ① Open goals → (If No causal links then flaw)

There is no action which produces the subgoal.

② Threat → No threat should be there

i.e an action threatens the causal link.

(shown in prev. pages)

⇒ An action A threatens the causal link $\langle a, P, b \rangle$

if C produces not of P ①
& happen after a & before b.
② ③

To resolve threat, one of 3 shouldn't happen.

⇒ Partial plan is a solution plan if it has no flaws

High lvl algo ⇒
① Start with intial plan
② Keep refining it
③ Do ② till no flaws.

Algo → PSP(π_0)

$\pi \leftarrow \pi_0$
While π has a flaw
Choose a flaw f
Let R(f) be set of resolvers for f
If R(f) = ϕ then
return FAIL
else
Choose $r \in R(f)$
$\pi \leftarrow$ Apply (r, π)
(flaw resolved)
Return π

⇒ PSP should be able to solve Sussman's anomaly.

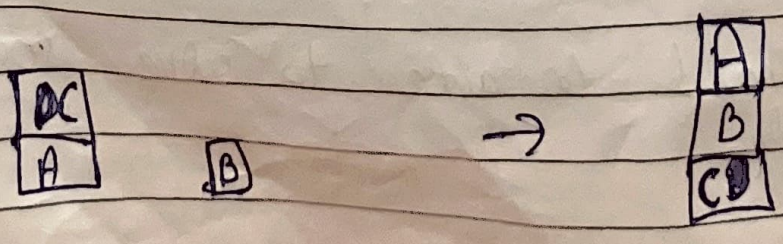
Page No.
Date

⇒ How to address openGoal flaw?
↳ 2 ways :-
① Existing action in the plan can supply a predicate.
② Add a new action.

How to resolve a threat :-
3 resolvers
① Separation → Add Binding Constraint
② Demotion
If A is a threat to $\langle A_1, P, A_2 \rangle$
then Add ordering $(A_2 < A)$
(A_2 happens before A)
③ Promotion
Add $(A_1 < A)$ to ordering set
(A_1 happens before A).

⇒ We have 5 resolvers :-
(2+3)
For open goal ⇒ 2 resolvers
↳ Existing Action
↳ New Action
For threat :-

Sussman's Anomaly



Page No.
Date

① threatened by
②

→ ordering link

⇒ ③ & ④ connected

~~Not~~

Actions

PickupA – holding A
UnstackCA – PutdownA – PickupB
StackBC
StackAB

⇒ From this we can make a linear plan (Using topological Sort)