Knowledge representation and reasoning
Lecture 19: Planning 2

Natasha Alechina
natasha.alechina@nottingham.ac.uk

Plan of the lecture

1. Recap: Planning problem
2. Forward and regression planning
3. Goal stack planning. Sussman anomaly

Planning domain

- Fluents or predicates which are used to describe preconditions and effects of actions
- For example, in the blocks world domain:
  - $Block(x)$: x is a block
  - $On(x, y)$: x is on y, where y is either another block or table
  - $Clear(x)$: there is nothing on top of block x
- Action schemas: available actions and their postconditions and effects. For example,

  ACTION: $moveToTable(b, x)$:
  PRECONDITION: $Block(b), Block(x), On(b, x), Clear(b)$
  EFFECT: $On(b, Table), Clear(x), \neg On(b, x)$

Example action schemas for blocks world

ACTION: $move(b, x, y)$:
PRECOND: $Block(b), Block(y), On(b, x), Clear(b), Clear(y), x \neq y$
EFFECT: $On(b, y), Clear(x), \neg On(b, x), \neg Clear(y)$

ACTION: $moveToTable(b, x)$:
PRECONDITION: $Block(b), Block(x), On(b, x), Clear(b)$
EFFECT: $On(b, Table), Clear(x), \neg On(b, x)$

Add and delete lists

- Given an action schema
  ACTION: a
  PRECONDITION: some literals
  EFFECT: $E_1, \ldots, E_m$
- $Add(a) = \{E \mid E$ is a positive literal in EFFECT$\}$ (positive effects of a)
- $Del(a) = \{P \mid E = \neg P$ where $E \in$ EFFECT$\}$ (atoms appearing with negation in the effect of a)

Add and delete list example

ACTION: $move(b, x, y)$:
PRECOND: $Block(b), Block(y), On(b, x), Clear(b), Clear(y), x \neq y$
EFFECT: $On(b, y), Clear(x), \neg On(b, x), \neg Clear(y)$

- $Add(move(b, x, y)) = \{On(b, y), Clear(x)\}$
- $Del(move(b, x, y)) = \{On(b, x), Clear(y)\}$

Planning problem

- Planning domain
- Objects (what can be substituted for variables)
- Initial state: a list of positive ground (no variables) literals, which are properties that hold in the state.
- Using Closed World Assumption, only describe what holds, and all the rest is assumed false
- Goal state: what literals should hold in the goal state. Can contain variables, which are assumed to be existentially quantifed.

Planning problem example

- planning domain as before
- objects: blocks A, B, C
- initial state: $\{Block(A), Block(B), Block(C), Clear(C), Clear(A), On(C, B), On(A, Table), On(B, Table)\}$. In this state, $On(A, Table)$ is true and $\neg On(C, Table)$ is true.
- Example of a non-ground goal: $On(x, B)$ as a goal means something should be on top of B; it is true in the initial state because x can be substituted for C.

Solving the planning problem

- Can be solved using search
- State space is not explicitly given; states are all possible sets of positive ground fluents
- Operators are actions; $do(s, a) = (s - Del(a)) \cup Add(a)$
- Plan = sequence of actions from initial state to a goal state, as in classical search

Forward or progressive planning

Algorithm from the textbook; S is current state, S' next (progressed) state

Input: a state and a goal
Output: a plan or fail.
ProgPlan[S,Goal] =
If Goal is satisfied in S, then return empty plan
For each operator *o* such that *precond*(*o*) is satisfied in S:
Let S' = S + add(o) - del(o)
. Let plan = ProgPlan[S',Goal]
. If plan $\neq$ fail, then return [act(o); plan]
End for
Return fail

Regression planning

Algorithm from the textbook; G is the goal, G' is regressed goal

Input: a state and a goal
Output: a plan or fail.
RegrPlan[S,Goal] =
If Goal is satisfied in S, then return empty plan
For each operator *o* such that $del(o) \cap Goal = \emptyset$
. Let Goal' = Goal + precond(o) - add(o)
. Let plan = RegrPlan[S,Goal']
. If plan $\neq$ fail, then return [plan; act(o)]
End for
Return fail

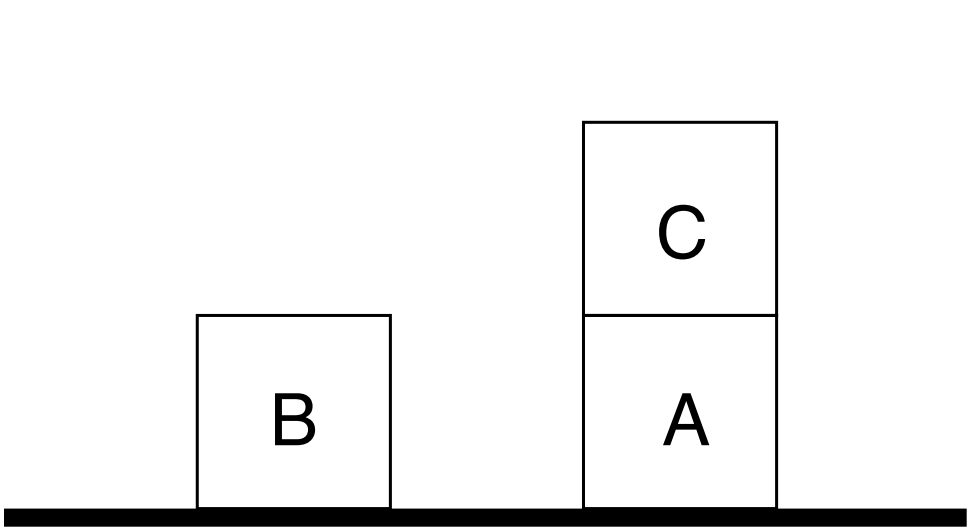
Goal stack planning

- One of the earlier planning algorithms called goal stack planning. It was used by STRIPS.
- We work backwards from the goal, looking for an operator which has one or more of the goal literals as one of its effects and then trying to satisfy the preconditions of the operator.
- The preconditions of the operator become subgoals that must be satisfied. We keep doing this until we reach the initial state.
- Goal stack planning uses a stack to hold goals and actions to satisfy the goals, and a knowledge base to hold the current state, action schemas and domain axioms
- Goal stack is like a node in a search tree; if there is a choice of action, we create branches

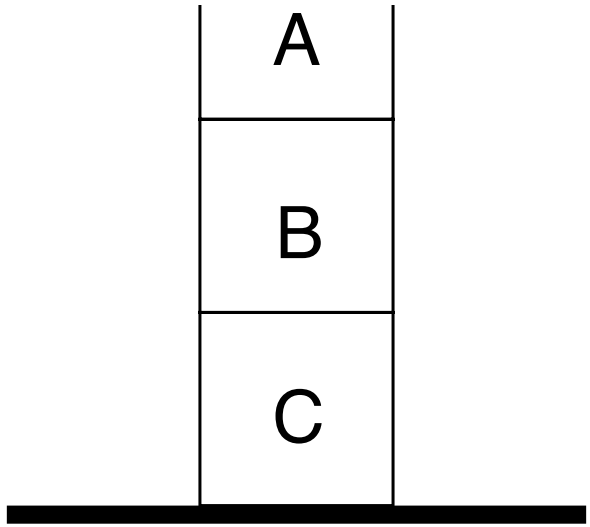
Goal stack planning pseudocode

- Push the original goal on the stack. Repeat until the stack is empty:
- If stack top is a compound goal, push its unsatisfied subgoals on the stack.
- If stack top is a single unsatisfied goal, replace it by an action that makes it satisfied and push the action's precondition on the stack.
- If stack top is an action, pop it from the stack, execute it and change the knowledge base by the action's effects.
- If stack top is a satisfied goal, pop it from the stack.

Goal stack planning example



Start



Goal

Goal stack planning 1

(I do pushing the subgoals at the same step as the compound goal.)
The order of subgoals is up to us; we could have put $On(B, C)$ on top

$$On(A, B)$$
$$On(B, C)$$
$$On(C, Table)$$
$$On(A, B) \wedge On(B, C) \wedge On(C, Table)$$

$S = \{On(C, A), On(A, Table), On(B, Table), Clear(B), Clear(C)\}$
plan = [ ]
The top of the stack is a single unsatisfied goal. We push the action which would achieve it, and its preconditions.

Goal stack planning 2

$On(A, Table)$
$Clear(B)$
$Clear(A)$
$On(A, Table) \wedge Clear(A) \wedge Clear(B)$
$move(A, x, B)$
$On(A, B)$
$On(B, C)$
$On(C, Table)$
$On(A, B) \wedge On(B, C) \wedge On(C, Table)$

$S = \{On(C, A), On(A, Table), On(B, Table), Clear(B), Clear(C)\}$
plan = [ ]
The top of the stack is a satisfied goal. We pop the stack (twice).

Goal stack planning 3

$$Clear(A)$$
$$On(A, Table) \wedge Clear(A) \wedge Clear(B)$$
$$move(A, Table, B)$$
$$On(A, B)$$
$$On(B, C)$$
$$On(C, Table)$$
$$On(A, B) \wedge On(B, C) \wedge On(C, Table)$$

$S = \{On(C, A), On(A, Table), On(B, Table), Clear(B), Clear(C)\}$
plan = [ ]
The top of the stack is an unsatisfied goal. We push the action which would achieve it, and its preconditions.

Goal stack planning 4

$On(C, A)$
$Clear(C)$
$On(C, A) \wedge Clear(C)$
$moveToTable(C, A)$
$Clear(A)$
$On(A, Table) \wedge Clear(A) \wedge Clear(B)$
$move(A, Table, B)$
$On(A, B)$
$On(B, C)$
$On(C, Table)$
$On(A, B) \wedge On(B, C) \wedge On(C, Table)$

$S = \{On(C, A), On(A, Table), On(B, Table), Clear(B), Clear(C)\}$
plan = [ ]
The top of the stack is a satisfied goal. We pop the stack (three times).

Goal stack planning 5

$moveToTable(C, A)$
$Clear(A)$
$On(A, Table) \wedge Clear(A) \wedge Clear(B)$
$move(A, Table, B)$
$On(A, B)$
$On(B, C)$
$On(C, Table)$
$On(A, B) \wedge On(B, C) \wedge On(C, Table)$

$S = \{On(C, A), On(A, Table), On(B, Table), Clear(B), Clear(C)\}$
plan = [ ]
The top of the stack is an action. We execute it, update the state with its effects, and add it to the plan.

Goal stack planning 6

$$Clear(A)$$
$$On(A, Table) \wedge Clear(A) \wedge Clear(B)$$
$$move(A, Table, B)$$
$$On(A, B)$$
$$On(B, C)$$
$$On(C, Table)$$
$$On(A, B) \wedge On(B, C) \wedge On(C, Table)$$

$S =$
$\{On(C, Table), On(A, Table), On(B, Table), Clear(A), Clear(B), Clear(C)\}$
plan = $[moveToTable(C, A)]$
The top of the stack is a satisfied goal. We pop the stack (twice).

Goal stack planning 7

$$move(A, Table, B)$$
$$On(A, B)$$
$$On(B, C)$$
$$On(C, Table)$$
$$On(A, B) \wedge On(B, C) \wedge On(C, Table)$$

$S =$
$\{On(C, Table), On(A, Table), On(B, Table), Clear(A), Clear(B), Clear(C)\}$
plan = $[moveToTable(C, A)]$
The top of the stack is an action. We execute it, update the state with its effects, and add it to the plan.

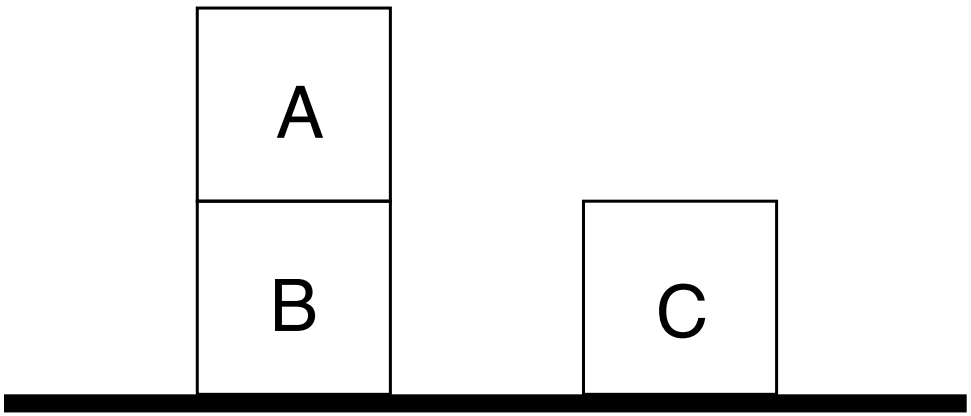
Goal stack planning 8

$On(A, B)$
$On(B, C)$
$On(C, Table)$
$On(A, B) \wedge On(B, C) \wedge On(C, Table)$

$S = \{On(C, Table), On(A, B), On(B, Table), Clear(A), Clear(C)\}$
plan = $[moveToTable(C, A), move(A, Table, B)]$
top of the stack is a satisfied goal

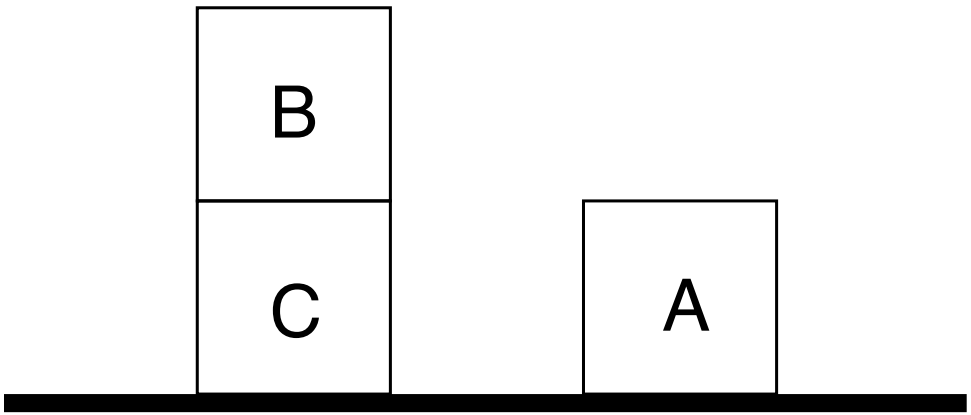
Goal stack planning 9a

the current state is



Goal stack planning 9b

If we follow the same process for the $On(B, C)$ goal, we end up in the state



Goal stack planning 10

Now we finally can move A on top of B, but the resulting plan is redundant:

$$moveToTable(C, A)$$
$$move(A, Table, B)$$
$$moveToTable(A, B)$$
$$move(B, Table, C)$$
$$move(A, Table, B)$$

There are techniques for ‘fixing’ inefficient plans (where something is done and then undone) but it is difficult in general (when it is not straight one after another)

Why this is an instructive example

- Sussman anomaly (called ‘anomaly’ because it seemed to make sense for a conjunctive goals to first achieve one goal and then achieve another goal, and then the complete goal would be achieved) is instructive,
- because it shows that although planning has the advantage over search in that states are factored and we can split the goal into subgoals, it is not straightforward
- achieving one goal ($On(A, B)$) destroys preconditions of an action which is necessary to achieve the other goal ($On(B, C)$), namely $Clear(B)$.
- Such interaction between actions is called clobbering.
- in the next lecture, I will talk about a solution to this problem (partial order planning)

Next lecture

- still planning (and SET and SEM)
- Partial order planning: Russell and Norvig, 3rd ed., 10.4.4.
- Goal stack planning was based on Elaine Rich and Kevin Knight, Artificial Intelligence textbook.