EECS 496: Sequential Decision Making

Soumya Ray

sray@case.edu

Office: Olin 516

Office hours : T 4-5:30 or by appointment

Announcements

- Exam TBA
  - In class, 75 minutes
  - Bring a calculator
- Possibility that one lecture in the upcoming two weeks will be canceled/rescheduled/given by a TA

Recap

- In the automated planning problem, an agent is given(i) _____ (ii) _______ and (iii) ________. It needs to find a ___ which is a _______, often also _____ some criterion.
- In classical planning, we assume the world is ____, ____, ____ and ___. Actions are ____.
- What is the “makespan”?
- We describe planning problems using ____ which take ____. These can be _____.
- States in STRIPS are _____ of ____ ___literals.
- What is the “closed world assumption”? Why is it useful?
- Goals in STRIPS are _____ of ____ ___literals.
- An action schema has an action ___ and ____, followed by lists of _____ and ____. The first contains a ____ of ____ literals. The second contains a _____ of literals.
- What are ADD and DELETE lists?
- What is the STRIPS assumption? Why is it useful?
- What are two kinds of planning algorithms?
- How do we formulate state space planning as a search problem?
- What is an admissible heuristic?
- We can formulate heuristics for planning through ____ plans or ____. ____ plans are obtained by _____ the ____ lists in ____ ____. ______ are obtained by considering a _____ _____ _____ at a time.
- Multiple admissible heuristics can be combined through the ____ function.

Today

- Automated Planning (Ch 10 R&N)

Backward State-Space Search

- Forward search has trouble with irrelevant actions
  - E.g., suppose there are 50 blocks on the table and I just want A on top of B
- This could be solved by backward search ("regression planning")
  - But this requires "inverting" the search operators (i.e. the actions)
  - Fortunately this is easy to do for STRIPS

Relevant Actions

- An action is relevant to a goal if its effects include at least one literal in the goal
- Suppose the goal is $On(A,B)$, then the only relevant actions are $Move(A,x,B)$

Backward State Space Search

- Start with the goal conjunction G (initial state)
- For each relevant action A (search operators), produce a new state by:
  - Deleting any literals in G that are added by A
  - Adding all precondition literals of A to G (unless they are already present)
- Goal test: A state which is satisfied by the initial state of the planning problem

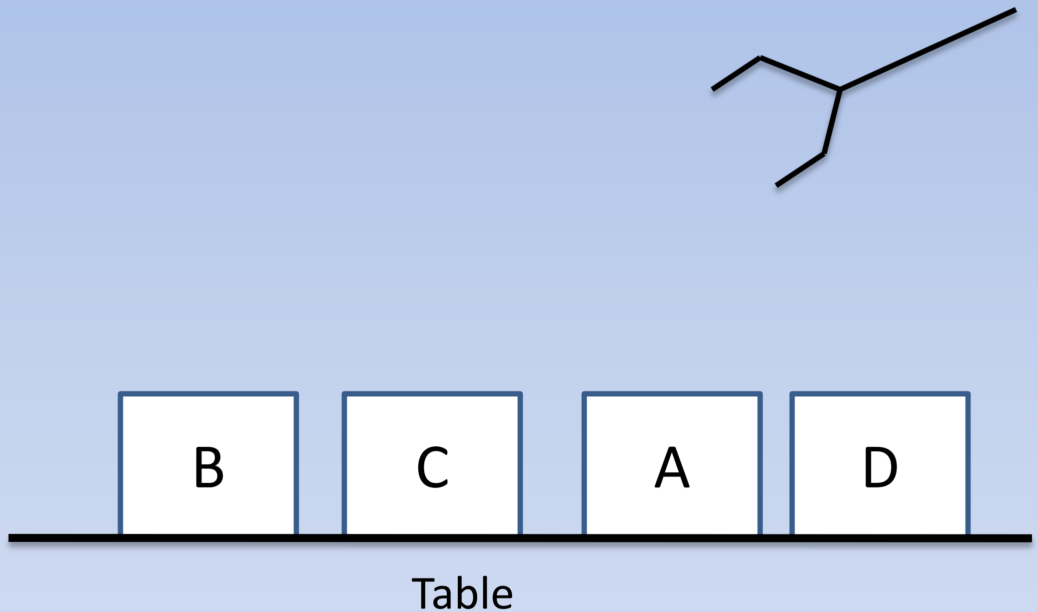
Example

- Goal: $On(A,B)$
- $Move(A,x,B)$
  - Preconditions: $On(A,x)$, $Clear(B)$, $Clear(A)$
  - Add List: $On(A,B)$, $Clear(x)$
  - Delete List: $On(A,x)$, $Clear(B)$
- Result: $On(A,C)$, $Clear(B)$, $Clear(A)$

Kinds of Search for Planning

- Search algorithms for classical planning fall into two categories
  - “State space planners”: States of the search problem are states of the world; search operators are actions of the world
  - “Plan space planners”: States of the search problem are partial plans; search operators are modifications to the current partial plan

Blocks World



Goal: $On(A,B) \wedge On(C,D)$
Solutions: {*Move(A, Table, B), Move(C,Table,D)*}, {*Move(C, Table, D), Move(A,Table,B)*}

Total Order Plans

- In a Total Order plan, every pair of actions A_1 and A_2 has a *temporal ordering constraint*
  - Either A_1 is done first, or A_2
  - Forward state space planners produce plans like this

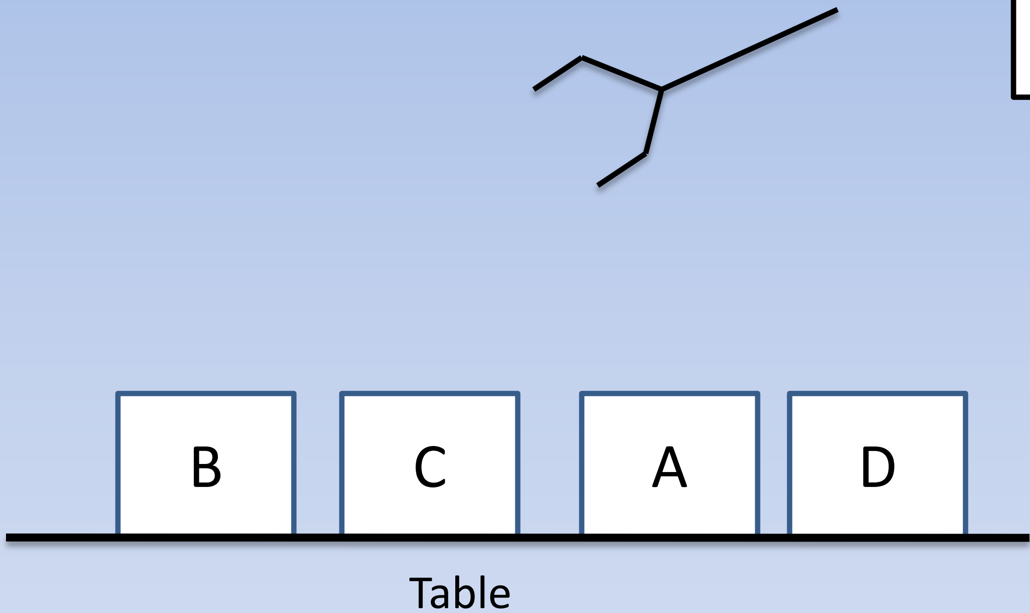
Partial Order Plans

- In many situations, actions do not have to be done in order
  - Might be trying to achieve unrelated things

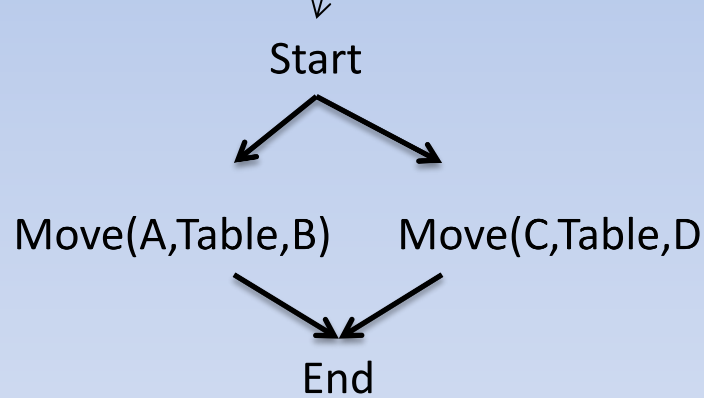
- This creates a *partial order* plan: a plan with some actions that have no temporal ordering constraints between them
  - i.e. there is some A_1, A_2 so that A_1 does not have to be completed before A_2 and A_2 does not have to be completed before A_1 for the plan to succeed

Blocks World

A dummy action with no preconditions and effect==initial state



Goal: *On(A,B),On(C,D)*



A dummy action with no effects and preconditions==goal

Partial Order Plans

- Represented as a set of actions and ordering constraints ($A < B$)
  - Ordering constraints can’t introduce cycles

- Partial Order plans have three advantages over total order plans
  - Action Parallelism
  - Flexibility when executing the plan
  - (Partial) Failure resilience

Finding a POP

- To find POPs, we will perform a *plan-space* search

- The states of the search space will be *incomplete POPs*, augmented with some bookkeeping information

States of the POP Search Space

- A state will have
  - The incomplete POP (list of actions and ordering constraints)
  - A list of *open conditions*
  - A list of *causal links*

Open Conditions

- A precondition of some action in the plan that is not currently the effect of some action in the plan
- The termination states of the search are those where this list is empty
- Initially, this is the goal, represented by the dummy action End

Causal Links

- Suppose an action B in the incomplete POP has an open precondition p
- Suppose the planner adds an action A to the POP that has p as an effect
- The *cause* for adding A is so it fulfils p for B
- This is indicated in the POP by adding a *causal link*, denoted $A \xrightarrow{p} B$

Conflict and Threats

- Suppose the planner has just added A and a causal link $A \xrightarrow{p} B$
- Suppose there is an action C in the plan that has $\neg p$ as an effect and *could* be executed after A and before B
- We say that C *conflicts with*, or *threatens*, the causal link $A \xrightarrow{p} B$

Consistent Plans

- A *consistent plan* is a POP where:
  - the ordering constraints have no cycles and
  - there are no conflicts with causal links

- So a solution is a consistent plan with no open conditions

POP Search Algorithm

- Initial State: Plan=*(Start, End, Start< End)*, open conditions=preconditions of *End* (goal), no causal links

- Search operators: Pick an open condition, and an action to satisfy it; generate next state (a new, consistent, possibly incomplete POP)

- Goal test: empty list of open conditions

Generating the Next State

- Suppose we pick open condition p to satisfy, using action A
  – p must have been a precondition of some action B
  – A might be an action already in the plan

Generating the Next State

- To generate the next state:
  - Remove p from list of open conditions
  - Add A to list of actions in POP (if not using existing)
  - Add ordering constraints $Start < A$, $A < B$, $A < End$
  - Add A's preconditions to list of open conditions
  - Add a causal link $A \xrightarrow{p} B$
  - Resolve conflicts/threats if any

Conflict Resolution

- Conflicts can arise between
  - The new causal link and existing actions
  - The new action and existing causal links

- To resolve a conflict, add ordering constraints
  - Suppose C conflicts with $A \xrightarrow{p} B$
  - Then add either $C < A$ or $B < C$ (assuming no cycles)

Example

- Init: *At(Flat,Axle),At(Spare,Trunk)*
- Goal: *At(Spare,Axle)*
- *Remove(Spare,Trunk)*
  - PRE: *At(Spare,Trunk)*
  - EFF: ¬*At(Spare,Trunk), At(Spare,Ground)*
- *Remove(Flat,Axle)*
  - PRE: *At(Flat,Axle)*
  - EFF: ¬*At(Flat,Axle), At(Flat,Ground)*
- *PutOn(Spare,Axle)*
  - PRE: *At(Spare, Ground),* ¬*At(Flat,Axle)*
  - EFF: ¬*At(Spare, Ground), At(Spare,Axle)*
- *LeaveOvernight*
  - PRE:
  - EFF: ¬*At(Spare,Trunk),* ¬*At(Spare, Ground),* ¬*At(Spare,Axle),* ¬*At(Flat,Axle),* ¬*At(Flat,Ground)*