EECS 391: Introduction to AI

Soumya Ray

Website: http://engr.case.edu/ray_soumya/eecs391_sp17/

Email: sray@case.edu

Office: Olin 516

Office hours: M 12:30-2pm

Automated Planning (Ch 10)

- Consider again a situation where an agent has to carry out a sequence of actions to achieve a goal

- Suppose the agent starts off with detailed, *structured* knowledge of the world
  - Could we take advantage of this?

The Planning Problem

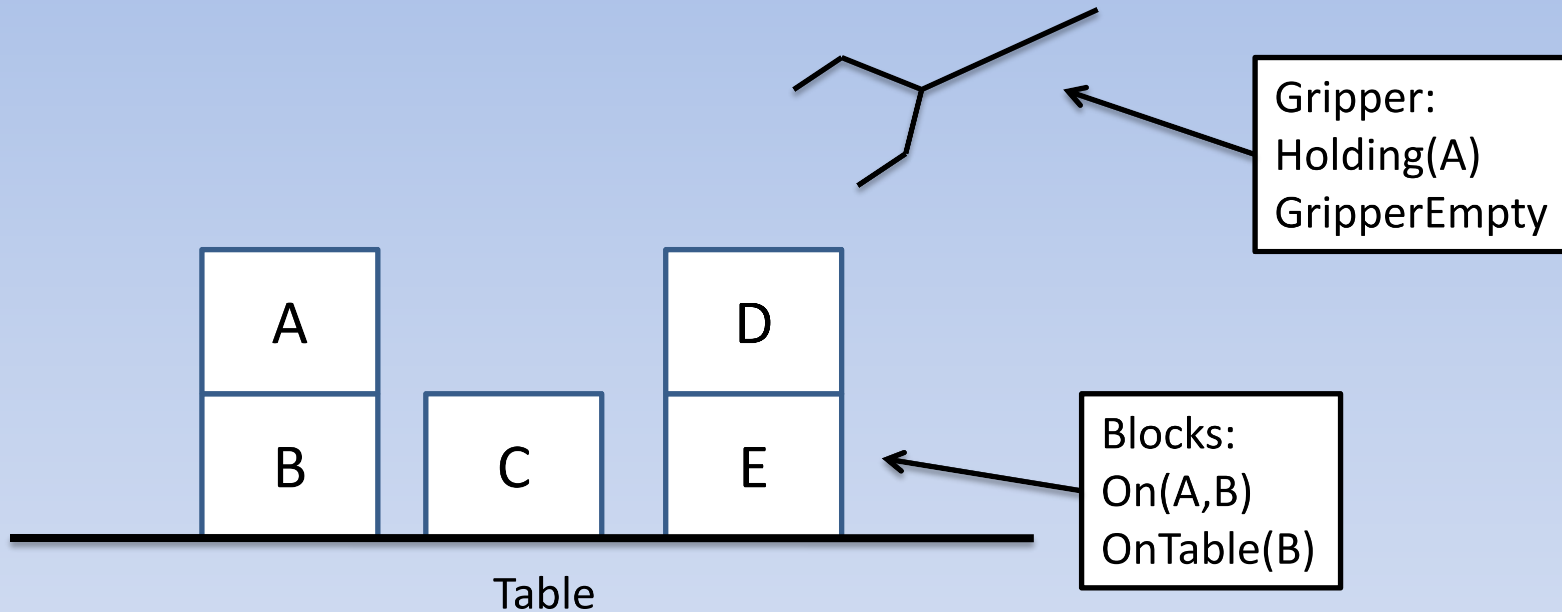
- Given:
  - An initial state of the world, described as a set of logical facts
  - A set of goal states, described as a set of logical facts
  - A set of actions, also described in logic

- Find a sequence of actions that will move the world from the initial state to the final state
  - This sequence is called a *plan*
  - Often also try to optimize some criteria

“Classical” Planning

- We’ll study planning algorithms designed to work when the world is:
  - Deterministic
  - Static
  - Fully observable
  - Actions are instantaneous
- These restrictions can be relaxed (more or less)

Blocks World



Task: Starting with initial configuration of blocks,
produce a desired goal configuration by moving blocks around.

Situation Calculus (Chapter 10.3)

- It is natural to think of using full FOL to encode states of the world and actions
  - Then use general FOL inference as planner

- People developed a general method for encoding states and actions based on FOL
  - Called the “Situation Calculus”

Situation Calculus

- A "*situation*" is a logical term that summarizes the current state of the world (state + time index)
- In each situation, the agent can take an action (another logical term), to get a new situation
- *Fluents* are functions or predicates that vary from one situation to the next

Example

$At(Agent, [1,1], S_0)$

$\neg Holding(Agent, Gold, S_0)$

$At(Agent,x,s) \wedge Adjacent(x,y) \Rightarrow Act(Go(x,y),s)$

$Result([],s)=s$

$Result([Act(a),Act(b)],s)=Result(Act(b),Result(Act(a),s))$

$Act(Go(x,y),s) \Rightarrow At(Agent,y,Result(Go(x,y),s))$

Issues with Situation Calculus

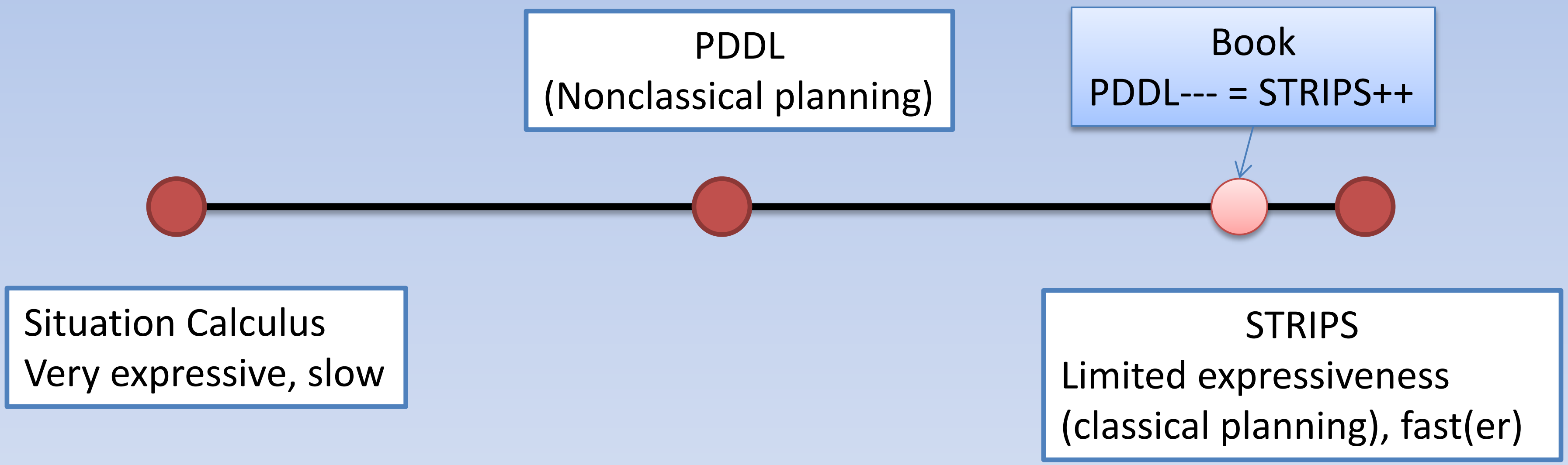
- SC is appealing because no special algorithms are needed for planning
  - Given an SC knowledge base, query “Is there a sequence of actions leading to a situation where the goal holds?”
  - Apply resolution
- But this is very slow, even for small planning problems
- So specialized fragments of FOL have been developed to represent planning problems instead

Representing a Planning Problem

- For classical planning, one fragment of FOL that is used is called STRIPS ("Stanford Research Institute Problem Solver")

- States, actions and goals will be represented in this language
  – Then we'll see planning algorithms (which are inference algorithms in disguise) that find plans in this language

Representing a Planning Problem



Representing States in STRIPS

- States in STRIPS are conjunctions of unnegated, ground, function-free literals
  - All conditions that hold in that state
  - *Block(A), Block(B), On(A,B), On(B, Table), GripperEmpty*
  - The "Closed World Assumption" is used

Closed World Assumption (CWA)

- Anything that is not explicitly listed is false
  - No “unknown” variables

Representing Goals in STRIPS

- Goals are conjunctions of unnegated, ground, function-free literals
- Goals may not fully determine a state of the world
  - In this case, the goal is any state where these literals hold
- Example: $On(A,E) \wedge On(B,D)$

Representing Actions in STRIPS

- Want to represent an action of picking up a block from the table

Pickup_from_Table(x)

Preconditions: *Block(x), GripperEmpty, Clear(x), On(x,Table)*

Add List: *Holding(x)*

Delete List: *GripperEmpty, On(x,Table)*

"Applicability": action can be used at a state iff its preconditions are satisfied

Representing Actions in STRIPS

- An "action schema" represents a non-ground action using three parts:
  - The action name and parameter list
  - The preconditions: a list of unnegated function-free (non-ground) literals. Any variables in this list are parameters to the action.
  - The effects: a list of function-free literals describing how the state changes.

Add and Delete Lists

- Often, the unnegated literals in the action effects are collected into an "ADD" list, and the negated literals are collected into a "DELETE" list
  - Idea: Starting with initial state, to get result of applying action, add the literals in ADD list and delete the literals in the DELETE list

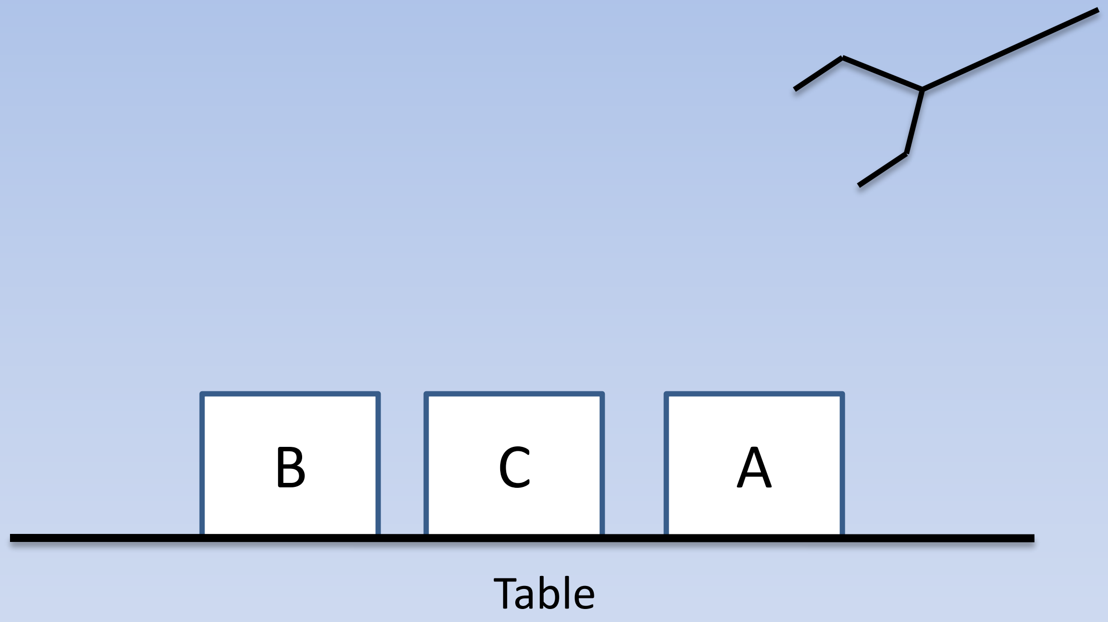
The STRIPS assumption

- Every possible effect of actions are listed
  - i.e., if a literal does not appear in the effects list, it is unchanged in the resulting state
  - Together with CWA, solves the “frame problem” in situation calculus

Restrictions in STRIPS

- States are described by unnegated ground function-free literals
- CWA
- Ground conjunctive goals
- Conjunctive effects of actions
- No equality

Example: Blocks World



Example

- Init($On(A,Table) \wedge On(B,Table) \wedge On(C,Table) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge$

$Clear(A) \wedge Clear(B) \wedge Clear(C) \wedge GripperEmpty$)

- Goal($On(A,B)$)
- Action($MoveToTable(b,x)$,
  - Preconditions($On(b,x) \wedge Clear(b) \wedge Block(b) \wedge Block(x) \wedge GripperEmpty$)
  - AddEffects($On(b,Table) \wedge Clear(x)$)
  - DelEffects($On(b,x)$)

Planning Algorithms

- Given a STRIPS representation of a classical planning problem, how do we solve it?
  - Since the world is static, deterministic, fully observable, we could use search
  - Remember that in this case, the search algorithm is actually performing logical inference

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

Forward State-Space Search

- "Progression" planning

- Setup:
  - States=world states (in STRIPS)
  - Initial state=given
  - Operators=*applicable* actions (in STRIPS)
  - Goal test=given (in STRIPS)
  - Operator costs=unit (minimize number of actions)

Makespan

- Typically, in classical planning, we are interested in minimizing the *duration* of the plan
  - Equivalent to the number of actions in the current setup
  - This is called the *makespan*

- Nonclassical planning allows arbitrary plan metrics to be minimized

Forward State-Space Search

- We could apply any search algorithm, e.g. A*

- The key differences are:
  – Only applicable actions need to be explored at a state
  – Getting the next state is done through the STRIPS specification of states and actions
  – Heuristics are based on planning ideas

Search Heuristics

- From any state, want to estimate the number of actions to search termination admissibly

- Two possibilities:
  - Relax the planning problem
  - Consider subproblems

Relaxed Plans

- There are different ways to arrive at a less constrained planning problem

- One way is to remove all DELETE effects from STRIPS actions
  - This is admissible (why?)
  - To estimate this cost, need to run an internal planning loop; but this is usually very fast

Subproblems

- The goal is a conjunction of literals

- We can generate subproblems by just considering a single literal at a time
  - "Subgoal Independence" (admissible)

- Combine with max, as usual

Total Order Plans

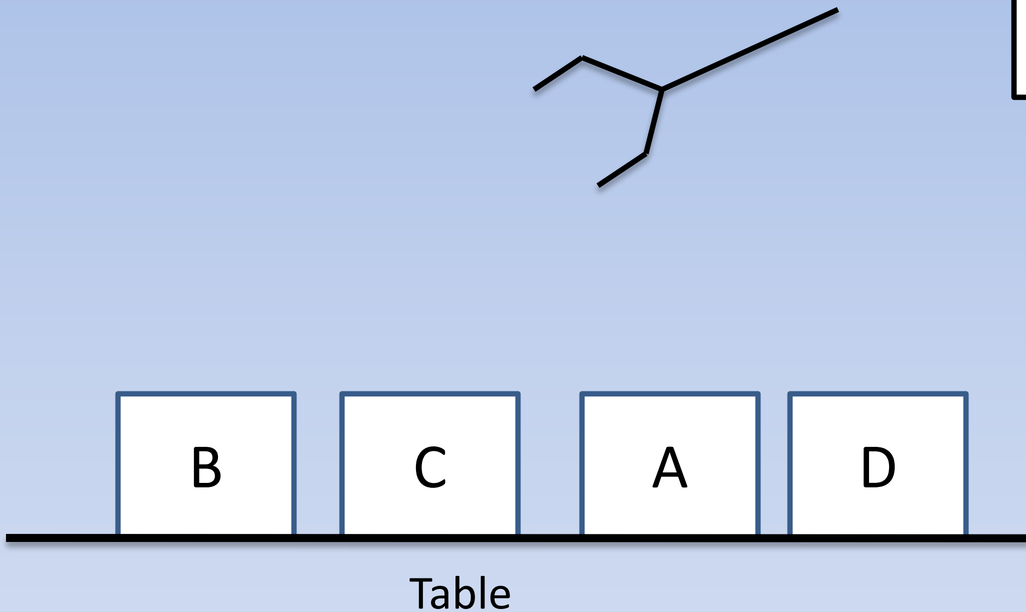
- In a Total Order plan, every pair of actions A_1 and A_2 has a *temporal ordering constraint*
  – Either A_1 is done first, or A_2
  – Forward state space planners produce plans like this

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



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