Today's Lecture

- Planning

Planning: general approach

- Use a (restrictive) formal language to describe problems and goals.
  - Why restrictive? More precision and fewer states to search
- Have a goal state specification and an initial state.
- Use a special-purpose **planner** to search for a solution.

Basic formalism

- Basic logical formalism derived from STRIPS.

- State variables determine what actions can or should be taken: in this context they are **conditions**
  - Shoe_untied()
  - Door_open(MC)

- An operator (remember those?) is now a **triple**

Preconditions

Additions

Deletions

Going forwards

- All state variables are **true** or **false**,
  but some may not be defined at a certain point on our

State Progression.

A planner based on this is a progression planner.

Idea:

In a state S,
Can apply operator $\underline{X}$=(P,A,D).
Leads to new state T

$$T = f_{\underline{X}}(S) = (S\text{-}D) \cup A$$

Constancy

- Important caveat

- When we go from one state to another,
  we assume that the
  only changes were those that resulted explicitly from the Additions and Deletions.

Given this assumption, the operator X computes the *strongest provable postconditions.*
In reality, even more might be deleted.

Aside: FOL with time

- One approach is a variation of first-order logic called **situation calculus** [McCarthy].
  - **Events** take place at specific times.
  - Some predicates are **fluents** and only apply for certain ranges in time.
  - A **situation** is a temporal interval over which all the predicates remain fixed.
  - This material from Ch. 6 will largely be skipped. We wil cover, or have covered, 6.6 on in class.
  - Please read chapter 6 unto the first page of 6.2, & the first 2 pages of 6.5

Going backwards

- Remember backwards chaining?

- State at the goal G.
- Assuming the deletions aren't there for some operator $\underline{X}$
  - Why?
- Can chain backwards by adding what would have been deleted and removing what would have been added

$$S = f^{-1}_{\underline{X}}(G) = (G\text{-}A) \cup D$$

```
Maybe we added too much (with D),
      or deleted too little?
```

Means/ends analysis

- How can we get from initial to final?
  - Assume the states and operators are given.
  - What's the right path? How to we measure distance?

- Means/ends analysis assumes we simply reduce the number of things that make our current state different from out goal.

STRIPS

- STRIPS is an old planning language
    - STanford Research Institute Problem Solver.
  - Less expressive than situation calculus

  - Initial state:

    At(office) & NOT(Have(Video)) & Have(Cash) & Have(Uncooked-kernels)

  - Goal state

    At(Home) & Have(Video) & Have(Cooked-Popcorn)

Schemas

- Basic operators assume a complete specification of the state in which they are applied.

- This can be tedious
  - An **operator schema** is a "generic" operator that has variables in it
    - Related to axiom schemas
    - Related to unification in logic (e.g. prolog)

E.g.

Tie_shoes(h), *Tie_necktie(h)*,
Tie_boat_rope(h), *Tie_straightjacket(h)*

might all be abstracted by

Tie_object(X,h)

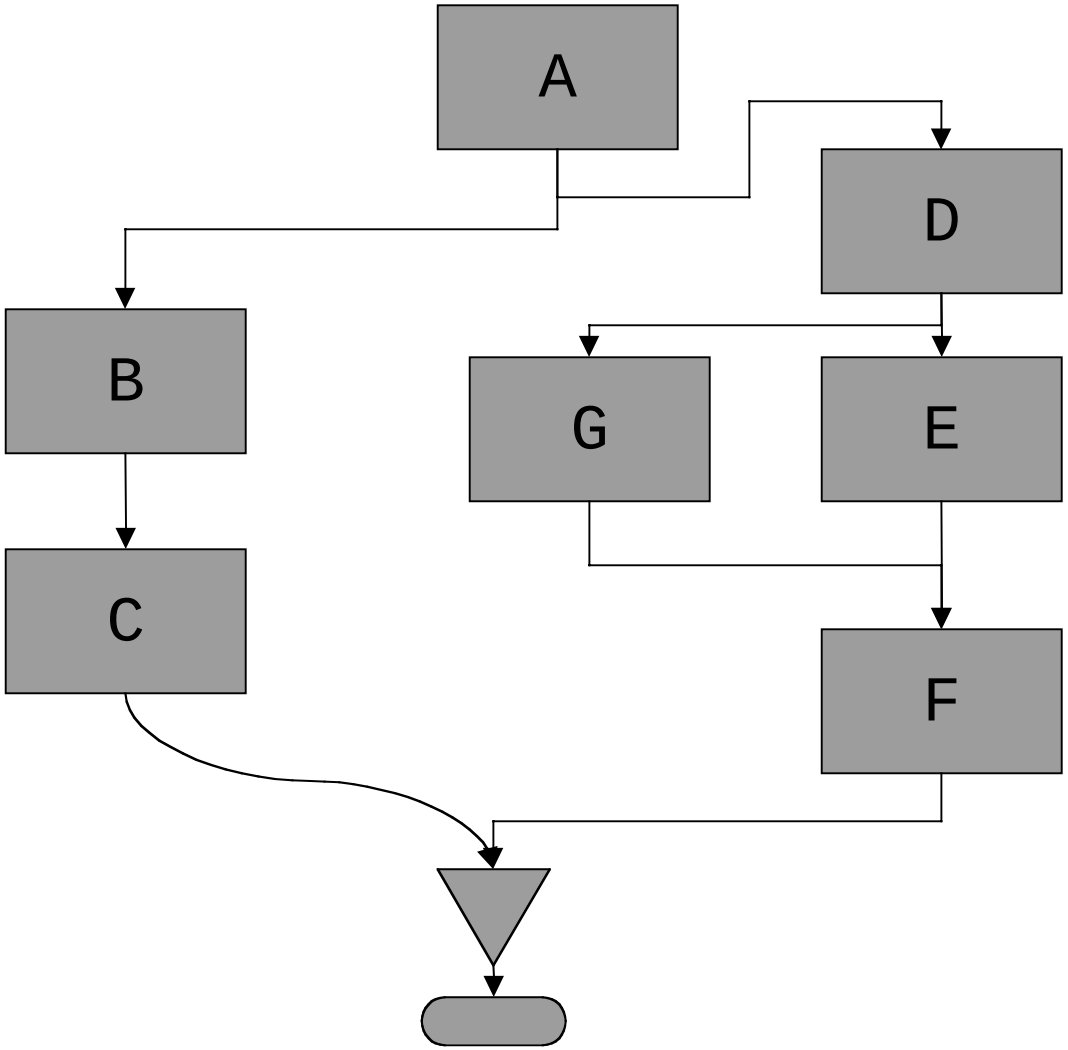
Least Commitment Planning

- When we formulate a plan intuitively, we often think of doing things in a specific sequence *even when the sequencing is arbitrary*.
  - This may not be wise.

- This can leads to re-shuffling actions...
  which is undesirable.

Generate plans such that we have sets of applicable actions, but we don't ***order*** the actions unless there is something (conditions) that demands it.

Partially ordered plan



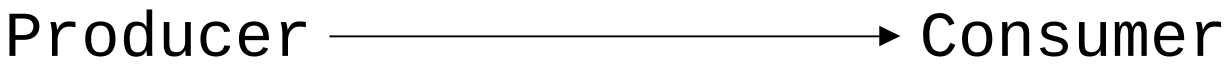
Terminology

- *Constraints* on sequencing, *requirements* for operators, *links* relating operators, *conflicts* between operators in a given plan.

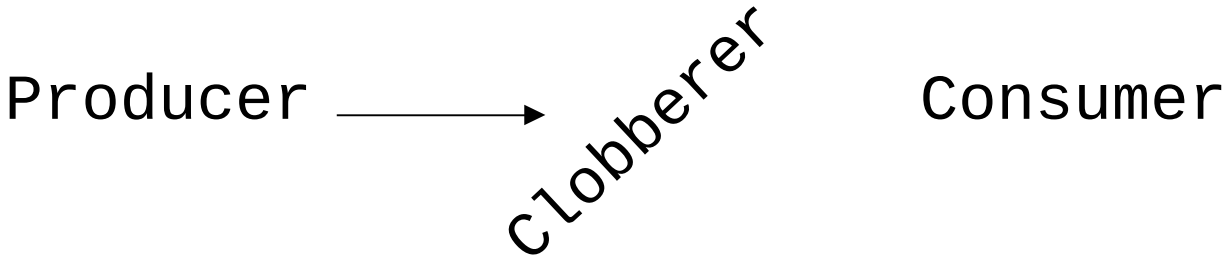
For a plan:

- Sound
  - Plan steps obey constraints on sequencing
  - Successful
- Systematic
  - Doesn't "waste" effort
- Complete
  - Generates a plan if one exists.
    - Still may not terminate (cf. Halting problem)
- Plan **refinement**
  - Improvement of an existing plan to make it better meet the constraints

Links & Conflicts



A conflict involves a link, & a step that messes it up.



Refinement

Fix conflicts by creating a new one from an old one.

- Keep old structures (links, producers, consumers, constraints) but add new constraints

- If there are conflicts, resolve them by adding constraints: move a *clobberer* before of after the link it's hitting.
  - (if you can).
- If there are no conflicts, satisfy an unfulfilled requirement.

Applications of planning

- Planning for Shakey the robot
  - Climb boxes
  - Push things
  - Move around
- Blocks world
  - Moving blocks
  - Piling them onto one another
  - Clearing the tops of chosen blocks

- Really doing this suggested we need **vision**!

Configuration Space Planning

CS-424 Gregory Dudek

Issues

CS-424 Gregory Dudek