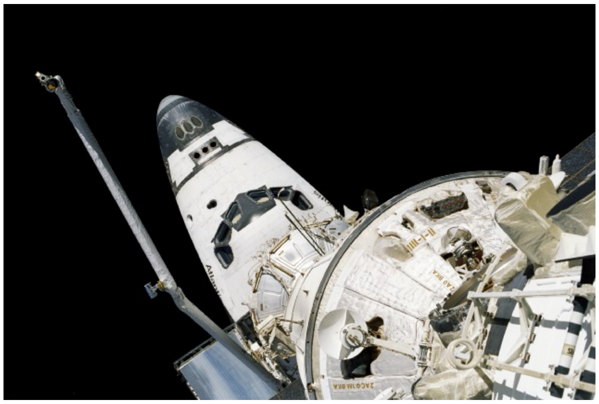
Planning

CPS 170

Ron Parr

An Example Planning Application



Space shuttle arm is currently controlled by a highly trained human.

Planning Application

- Remove human from the control loop
- Specific goals for system:
  - Rearrange items in cargo bay
  - Connect space station pieces
- Assuming mechanical engineering issues can be resolved:
  - Arm could work while astronauts sleep
  - Complicated training could be eliminated

Some Actual Planning Applications

- Used to fulfill mission objectives in Nasa's Deep Space One (Remote Agent)
  - Particularly important for space operations due to latency
  - Also used for rovers
- Aircraft assembly schedules
- Logistics for the U.S. Navy
- Observation schedules for Hubble space telescope
- Scheduling of operations in an Australian beer factory

Scheduling

- Many “planning” problems are scheduling problems
- Scheduling can be viewed as a generalization of the planning problem to include resource constraints
  - Time & Space
  - Money & Energy
- Many principles from regular planning generalize, but some extensions (not discussed here) are used

Characterizing Planning Problems

- Start state (group of states)
- Goal – almost always a group of states
- Actions
- Objective: Plan = A sequence of actions that is ***guaranteed*** to achieve the goal.
- Like everything else, can view planning as search…
- So, how is this different from search?

What makes planning special?

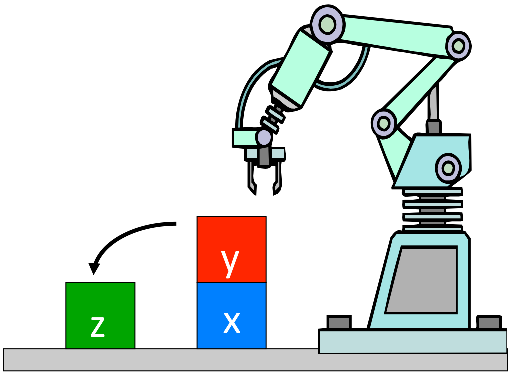
- States typically specified by a set of relations or propositions:
  - On(solar_panels, cargo_floor)
  - arm_broken
- Goal is almost always a set
  - Typically care about a small number of things:
    - at(Ron, airport),
    - parked_in(X, car_of(Ron))
    - airport_parking_stall(X)
  - Many things are irrelevant
    - parked_in(Y, car_of(Bill))
    - adjacent(X,Y)
- Branching factor is large

Planning Algorithms

- Extremely active and rapidly changing area
- Annual competitions pit different algorithms against each other on suites of challenge problems
- Algorithms compete in different categories
  - General vs. Domain specific
  - Optimal vs. Satisficing
- No clearly superior method has emerged, though there are trends

Planning With Logic/Theorem Proving

- Need to describe effects of actions with logic
- Ask for the existence of plans that achieve our goals
- Challenge: Talking about dynamic situations in logic



Situations

- Recall that we can't have contradictions in our knowledge base – OTW, can prove anything
- Need to index our claims about the world with time in some way (otherwise changes would create contradictions)
- Add an extra argument onto every predicate indicating when things are true:
  - on(table, z, s)
  - on(x, y, s)
- result(s,a) = result of doing a in s
- (result(s,a) = result(s',a,)) iff ((s=s') AND (a=a'))

Describing Actions

- Let's move A from B to C
- applicable(move(A,B,C,S)) :-
  - on(A,B,S)
  - clear(C,S)
- S'=result(move(A,B,C,S), S) :-
  - applicable(move(A,B,C,S))
  - clear(B,S')
  - on(A,C,S')

Successor State Axioms

- Action descriptions tell us what has changed, but how do we say what persists?
- This is the "frame problem"
- Successor state axioms:
  - On(C,D,result(A,S)) iff applicable(A,S) AND
    - On(C,D,S), A!=move, OR
    - On(C,D,S), A=move(A,B,C), C!=A, D!=B, OR
    - A=move(C,E,D)
- Need one of these for every proposition!

Finding the Plan

- Assume we have:
  - Descriptions of all actions
  - Successor-state axioms
  - Description of the initial state (situation)
- Q: How do we find the plan?
- A: Ask theorem prover if there exists a situation in which the goal is true!
- Theorem prover will return plan as a binding: result(move(X,Table,(result(move(Y,X,Z,S))

Planning via Theorem Proving: A Good Idea?

- Pros:
  - Very general
  - Very powerful representation
  - Access to theorem proving infrastructure
- Cons:
  - Awkward representation (unless you are a logician)
  - Slow in practice (price of generality)

Overcoming Limitations of Planning via Theorem Proving

- Simplify the representation
- Avoid successor state axioms
- Avoid generality of full, first order logic in hopes of allowing faster, special purpose algorithms for planning

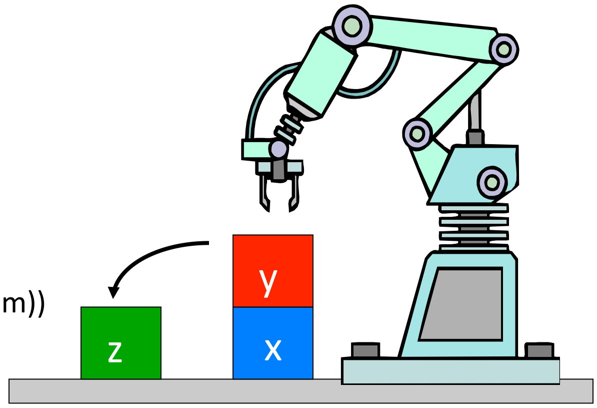
PDDL

- Actions have a set of preconditions and effects
- Think of the world as a database
  - Preconditions specify what must be true in the database for the action to be applied
  - Effects specify which things will be changed in the database if the action is taken

- NB: PDDL supersedes an earlier, similar representation called STRIPS

move(obj,from,to)

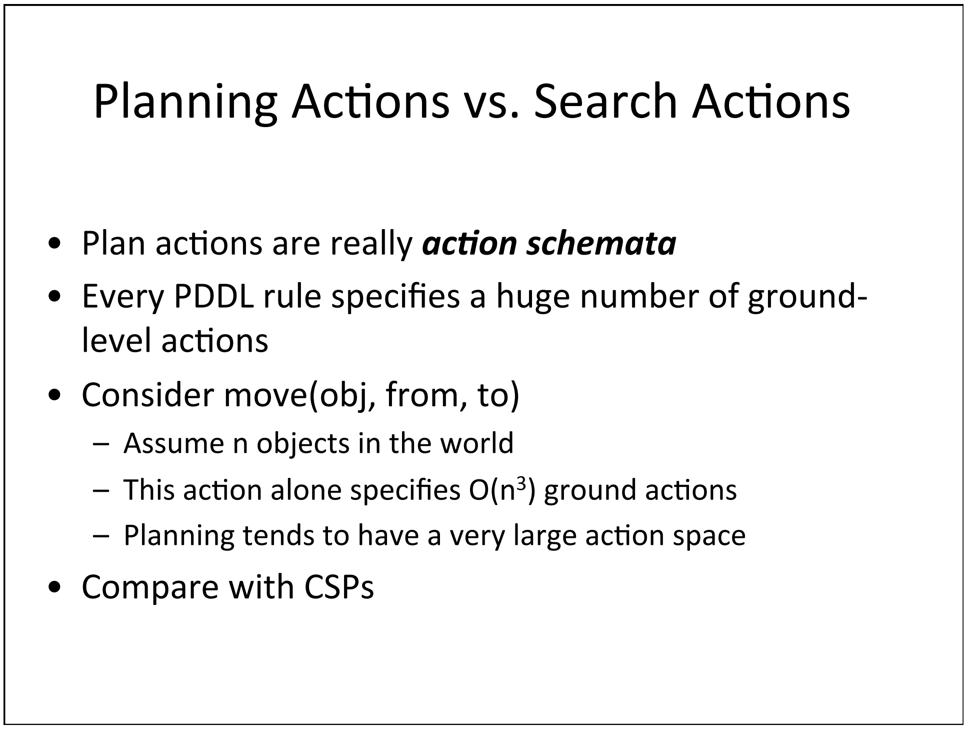
- Preconditions
  - clear(obj)
  - on(obj,from)
  - clear(to)
- Effects
  - on(obj,to)
  - clear(from)
  - not(on(obj,from))
  - not(clear(to))



move(y,x,z)

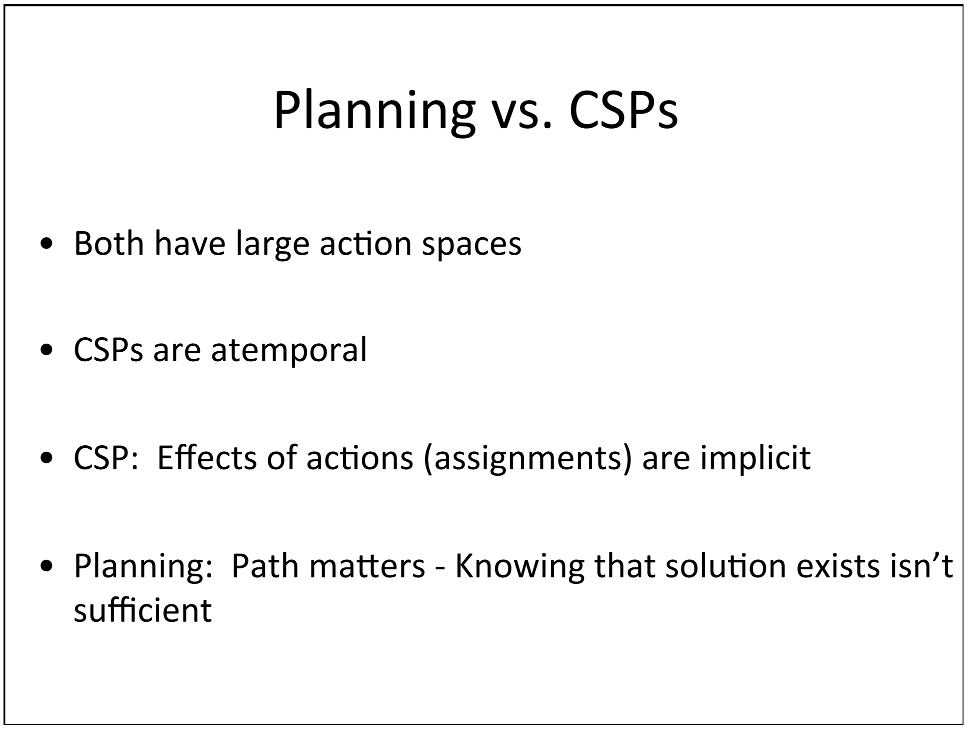
Limitations of PDDL

- Assumes that a small number of things change with each action
  - Dominoes
  - Pulling out the bottom block from a stack
- Preconditions and effects are conjunctions
- No quantification
- Closed world assumption (negation in effects only implemented as deletion)



Planning Actions vs. Search Actions

- Plan actions are really ***action schemata***
- Every PDDL rule specifies a huge number of ground-level actions
- Consider move(obj, from, to)
  - Assume n objects in the world
  - This action alone specifies O(n^3) ground actions
  - Planning tends to have a very large action space
- Compare with CSPs



Planning vs. CSPs

- Both have large action spaces
- CSPs are atemporal
- CSP: Effects of actions (assignments) are implicit
- Planning: Path matters - Knowing that solution exists isn't sufficient

How hard is planning?

- Planning is NP hard
- How can we prove this?
  - Use Planning to solve SAT
  - Any SAT instance can be converted to a planning problem in polynomial time
  - Polynomial time planning algorithm would imply polynomial time solution to SAT

Is planning NP-complete?

- NO!
- Consider the towers of Hanoi:
  - http://www.mazeworks.com/hanoi/index.htm
  - PDDL actions are the block moving actions
- Requires exponential number of moves
- Planning is actually PSPACE complete
- Planning with bounded plans is NP-complete

Should plan size worry us?

- What if you have a problem with an exponential length solution?
- Impractical to execute (or even write down) the solution, so maybe we shouldn't worry
- Sometimes this may just be an artifact of our action representation
  - Towers of Hanoi solution can be expressed as a simple recursive program
  - Nice if planner could find such programs

Planning Using Search

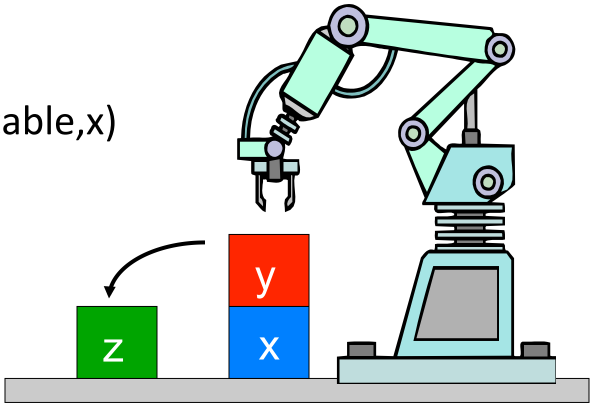
- Forward Search:
  - As with theorem proving, blind forward search is problematic because of the huge branching factor
  - Some success using this method with carefully chosen action pruning heuristics (not covered in class)
- Backward Search:
  - As with theorem proving, tends to focus search on relevant terms
  - Called "Goal Regression" in the planning context

Goal Regression

- Goal regression is a form of backward search from goals
- Basic principle goes back to Aristotle
- Embodied in earliest AI systems
  - GPS: General Problem Solver by Newell & Simon
- Cognitively plausible
- Idea:
  - Pick actions that achieve (some of) your goal
  - Make preconditions of these actions your new goal
  - Repeat until the goal set is satisfied by start state
- Note: Similar to backward chaing in theorem proving

Goal Regression Example

Regress on(x,z)
through move(z,table,x)



New goal:
clear(x)

Goal: on(x,z)

Facts About Goal Regression

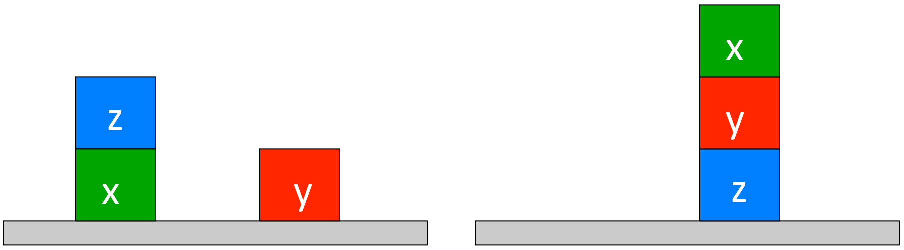
- Elegant solution to the problem of backward search from multiple goal states
  - In planning, goal state is usually a set of states
  - Does backward search at the level of state sets
- Goal regression is sound and complete
- Need to be careful to avoid endless loops on problems like Sussman anomaly (coming up)

Heuristics in planning

- Need heuristics for searching in planning, but…

- Planning problems tend to defy natural efforts to develop good heuristics:
  - Ignoring preconditions: Finding shortest path while ignoring preconditions is still an intractable problem
  - Ignoring deletions: Also intractable to find shortest path
  - (Above two difficulties mean that coming with an admissible heuristic is non-trivial.)
  - Counting number of conjuncts true (admissible but very weak)

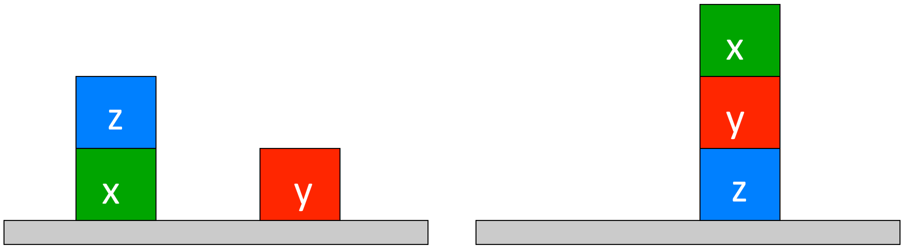
The Sussman Anomaly



Goal: on(x,y), on(y,z)

Problems with naïve subgoaling

- The number of conjuncts satisfied may not be a good heuristic
- Achieving individual conjuncts in isolation may actually make things harder
- Causes simple planners to go into loops



Planning Features & Challenges

- State space is very large
- Goals usually defined over state sets
- Very large, implicitly defined action space
- Difficult to come up with good heuristics
- Path (plan) usually matters

- We will see that ***plan graphs*** are a clever way of coming up with good heuristics for planners

Can our expertise in CSPs help?

- Can planning be reduced to CSPs?
- CSPs are a more restrictive language
- Need to consider bounded-length plans
  - In general, this isn't too much of a problem because extremely long plans are an indication that we need to reformulate the problem (Towers of Hanoi)

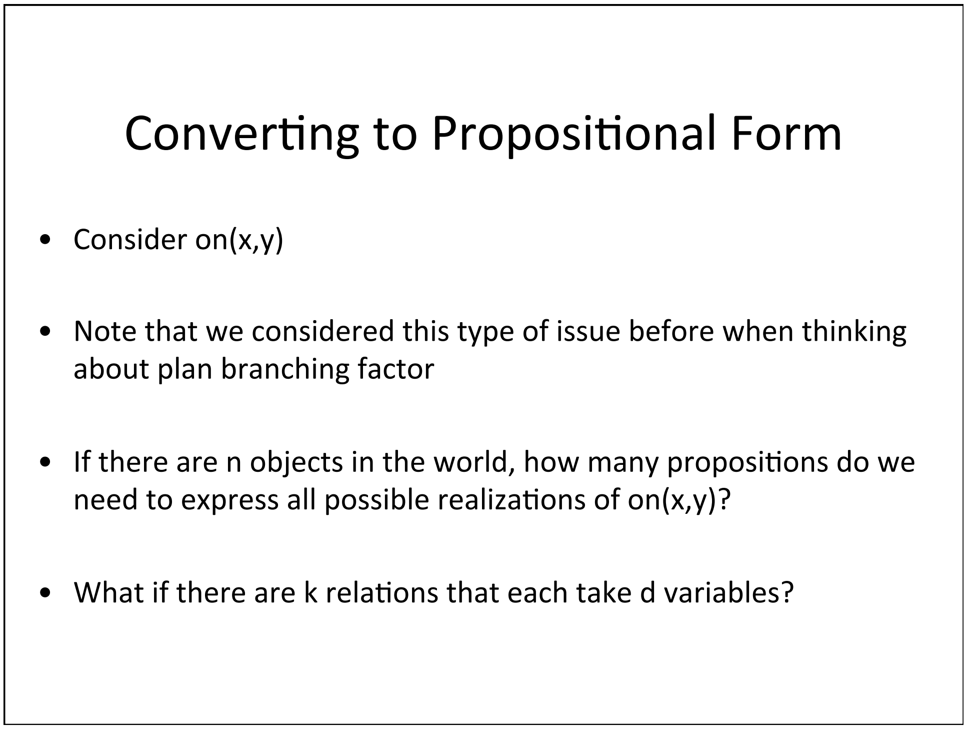
- Our hope: Solve plan as a CSP at let our CSP insights do the work for us (Doesn't quite work, but it helps…)

Formulating Planning as a CSP

- Introduce Action(a,i) (binary) to indicate if action a is taken at step i.
  - We introduce |Actions| x plan_length variables
- We also need to represent the statements in our database using proposition(p,i) (binary) to indicate the truth of proposition p at time I
  - This introduces |propositions| x plan_length variables
  - But there's a catch…

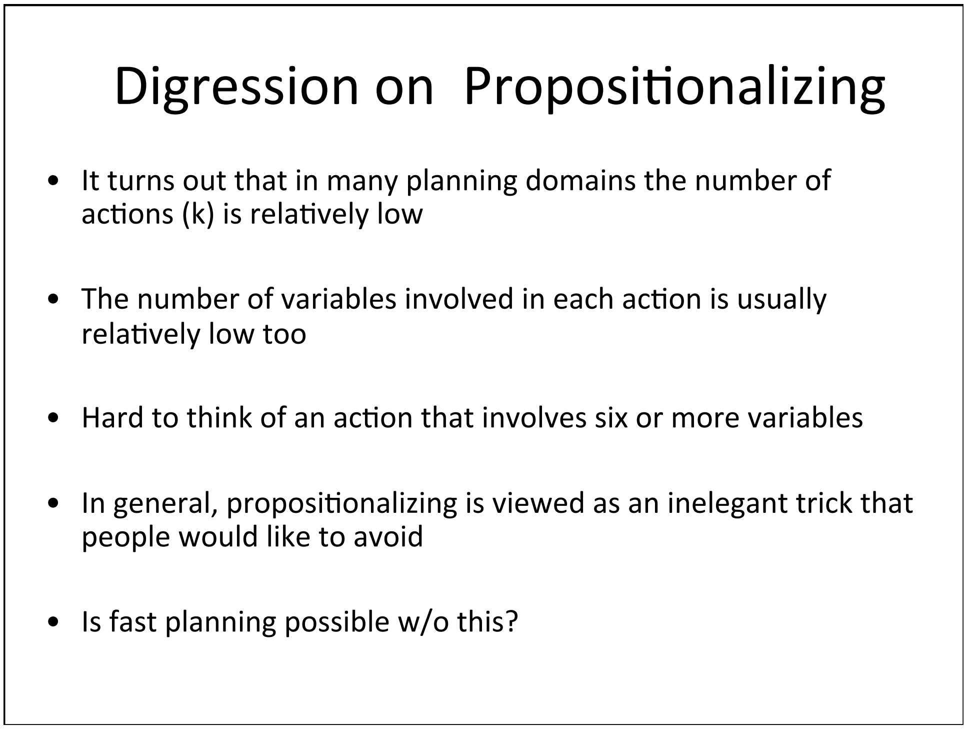
Propositionalizing

- Also called "grounding out"
- Recall that domain descriptions an actions involve relations:
  - on(x,table)
  - clear(x)
- Propositions don't take arguments
  - arm_broken



Converting to Propositional Form

- Consider on(x,y)
- Note that we considered this type of issue before when thinking about plan branching factor
- If there are n objects in the world, how many propositions do we need to express all possible realizations of on(x,y)?
- What if there are k relations that each take d variables?



Digression on Propositionalizing

- It turns out that in many planning domains the number of actions (k) is relatively low
- The number of variables involved in each action is usually relatively low too
- Hard to think of an action that involves six or more variables
- In general, propositionalizing is viewed as an inelegant trick that people would like to avoid
- Is fast planning possible w/o this?

Back to CSP formulation

- We now have action(move_x_y_z, i) = t iff we move x from y to z at time i.
- We also have proposition(on_y_z, i) = t iff y is on z at time i.
- Now we need to set up our constraints so that the problem is satisfiable iff there exists a plan

Plan CSP Constraints

- Actions must be sequential
  - For all a,a' not(action(a,i) and action(a',i))
  - Another quadratic factor!
- Actions' effects on the world. If action(a,i)=t
  - Proposition(p,i-1) = t for each p in preconditions
  - Proposition(p,i)=t for each true p in effects list
  - Proposition(p,i)=f for each false p in effects list
  - This is linear in the new action, proposition space

What’s Missing?

- We need to express that propositions persist
  - Proposition(p,i) = f unless
    - It was true in previous step and not deleted
    - It was false in previous step but not asserted
- We need to assert initial and final states
  - Easier than it sounds
  - We just set these variables to have the right values and the CSP does the rest

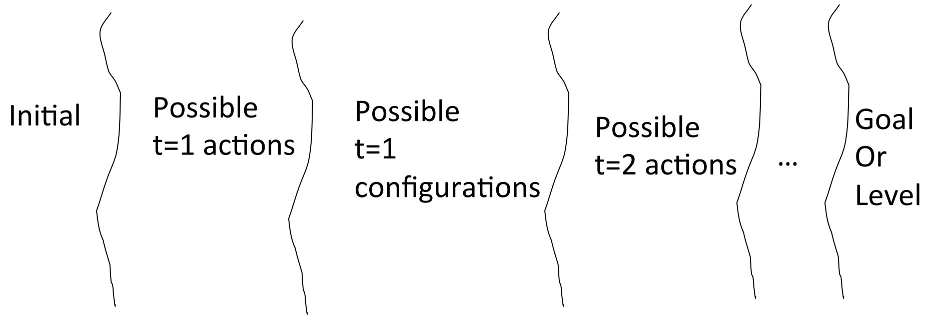
This works, but…

- The CSP is very large
- It is very highly connected
  - Variable elimination is hard
  - Hard to do k-consistency
- Turns a hard planning problem into a hard(er) CSP ☹

Plan Graphs

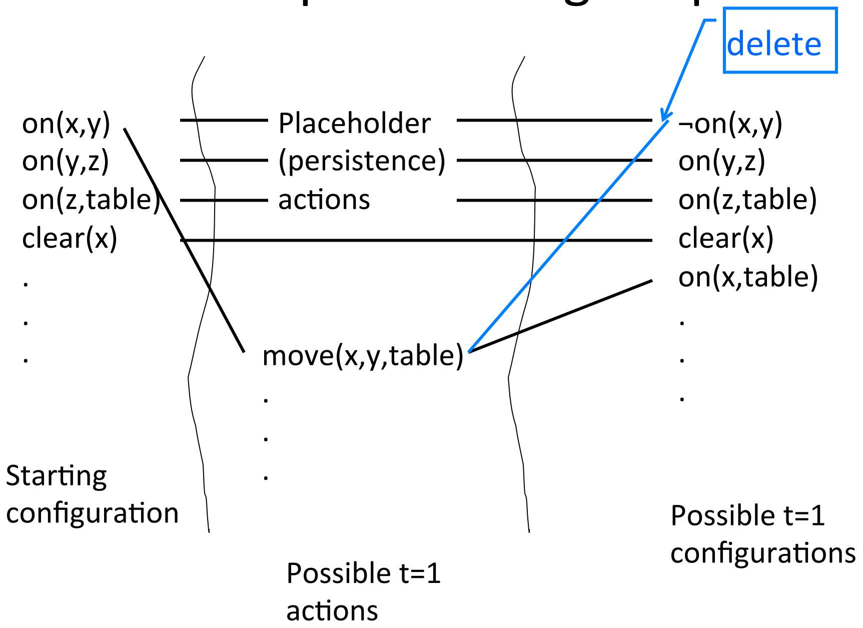
- High Level Idea:
  - Avoid constructing the exact CSP
  - Construct a sequence of simplified (trivially solvable) CSPs corresponding to different plan lengths
  - Never delete things
  - If plan of size k exists, then CSP of size k is satisfiable
  - Note is this ***if***, not ***iff***
- Why this is useful:
  - Did not have a good method of coming up with admissible heuristics for planning
  - If simplified CSPs are cheap to solve, then we have a reasonable, admissible heuristic

Plan Graph Form



Consider a particular world configuration c
Find the first phase containing all propositions in c
Distance to goal will be an admissable heuristic for forward search
and backward search.

Example Planning Graph



delete

on(x,y)
on(y,z)
on(z,table)
clear(x)
.
.
.

Placeholder
(persistence)
actions

move(x,y,table)
.
.
.

¬on(x,y)
on(y,z)
on(z,table)
clear(x)
on(x,table)
.
.
.

Starting configuration

Possible t=1 actions

Possible t=1 configurations

Facts About Planning Graphs

- Similar to CSP constraint graph
- The planning graph includes everything that *might* be true at a particular time
- Includes all actions that *might* be possible at a particular time
- Is a *relaxation* of the original problem

Why this is good

- Relaxations are a good way of developing admissable heuristics
- A major difficulty with planning is that we have trouble coming up with good heuristics
- Note that plan graphs can provide admissable heuristics for either direction (forward or regression [means/ends] search)

Why this isn’t good enough

- Produces fairly weak heuristics
- Propositions are never really deleted
- Can take many actions simultaneously
- The problem is now too relaxed
    - Need to figure out a way to use structure more effectively without losing
        - Problem independence
        - Clarity, speed
        - Admissability

Mutual Exclusion Between Actions

- Two real (non-persistence) actions can't be taken simultaneously; we mark these mutually exclusive
- Other types of mutual exclusion
  - Inconsistent effects/Interference
    - persist(on_x_y,1)
    - action(move_x_y_z, 1)
  - Competing needs
    - Precondition appears positive in one action
    - Appears negated in another

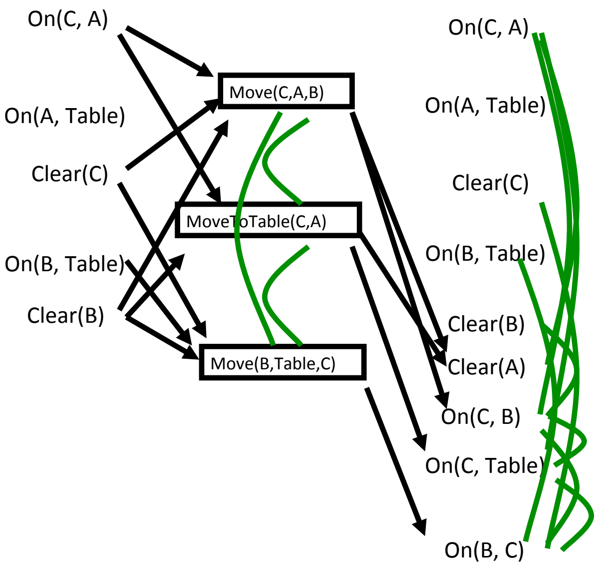
Extending graphs using mutex

For each planning phase:

1. Generate all actions with non-mutex preconditions
2. Mark as mutex all action/maintain pairs that conflict
3. Mark as mutex all action/action pairs with mutex preconds
4. Generate all potential propositions for next time step
5. Mark pairs of propositions that can only be generated by mutex actions as mutex

We now think of everything in terms of mutually compatible sets of propositions.

Plan Graphs with Mutex Constraints



On(C, A)

On(A, Table)

Clear(C)

On(B, Table)

Clear(B)

Move(C,A,B)

MoveToTable(C,A)

Move(B,Table,C)

On(C, A)

On(A, Table)

Clear(C)

On(B, Table)

Clear(B)

Clear(A)

On(C, B)

On(C, Table)

On(B, C)

- Each level has literals that “could be true” at that level
- Mutex (mutual exclusion) relations indicate incompatible actions/ literals

Slide courtesy Vince Conitzer

Plan Graphs with Mutex Constraints

- Extend forward until goal conjunctions appear non-mutex
- This is still a relaxation of the problem
- In essence, we have relaxed the original planning CSP so that we only worry about 2-consistency
- We still have an admissable heuristic
- For any configuration, we search for the earliest one in which the configuration propositions appear in non-mutex form

How do we use this?

- Original graphplan algorithm had a special planning algorithm that work with the plan graph
- Modern approaches primarily use the plan graph in conjunction with some kind of search
- Despite some apparent complexity, this turns out to be *much, much* cleaner, faster and easier to implement than planning algorithms from the 80s and early 90s

How well does it work?

- The initial graphplan algorithm was so much faster than competing algorithms it was hard to even compare them on the same scale.
- There is a web page devoted to graphplan:
  - http://www.cs.cmu.edu/~avrim/graphplan.html

Graphplan Summary

- Graphplan combines two concepts:
  - Constraint-based reasoning with a form of 2-consistency
  - Basic search
- More elaborate approaches are possible:
  - Add more complicated constraints to plan graph
  - Trade off: As plan graph becomes richer:
    - Heuristic values get closer to true plan length
    - Cost of building/using plan graph grows steeply
- Graphplan combines our knowledge of good search methods with our knowledge of good CSP methods

Other Approaches: SAT

- If we can convert planning to a CSP (kind of) and get some advantage from viewing it as a CSP, why not try converting to some other problem
- SATPlan converts (bounded length) planning problems to SAT problems
- Uses off-the-shelf SAT solvers
- As with plan graphs, this requires propositionalizing (grounding out) the problem

Interesting things about SATPlan

- This actually works pretty well for some domains
- Details of the transformation are somewhat tricky
- As with the CSP formulation, it tends to produce *very* large problem instances
- Can cause problems for domains with many items

Modern Planning Conclusion

- Fast planning algorithms seem to rely simple, fast underlying methods
- Ruling out bad things quickly seems to help
  - Heuristics used in SAT solvers (not covered here)
  - Constraint propagation in graphplan variants
- Still a very open area, not as clean as search/CSPs

What's Missing?

- As described, plans are "open loop"
- No provisions for:
  - Actions failing
  - Uncertainty about initial state
  - Observations

- Solutions:
  - Plan monitoring, replanning
  - Conformant/Sensorless planning
  - Contingency planning

Planning Under Uncertainty

- What if there is a probability distribution over possible outcomes?
  - Called: Planning under uncertainty, decision theoretic planning, Markov Decision Processes (MDPs)
  - Much more robust: Solution is a "universal plan", i.e., a plan for all possible outcomes (monitoring and replanning are implicit)
  - Much more difficult computationally
- What if observations are unreliable?
  - Called: "Partial Observability", Partially Observable MDPs (POMDPs)
  - Applications to medical diagnosis, defense
  - Way, way harder computationally