Planning

CSE 473
AIMA, 10.3 and 11

Overview

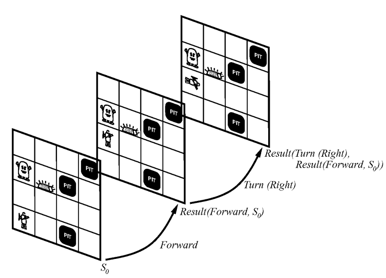
- FOL Planning in Situation Calculus
- Planning vs. Problem Solving
- STRIPS Formalism
- Partial Order Planning
- GraphPlan
- SATPlan

FOL Planning: Situation Calculus

- Situations: Logical description of world at some point in time
  - Result(a,s) returns next state / situation
- Fluents: Functions and predicates that change over time
  - Holding(G_1, S_4)
- Atemporal: Static functions and predicates
  - Gold(G_1)

Situation Calculus

- Result([], s) = s
- Result([a|seq], s) = Result(seq, Result(a, s))



Situation Calculus

- Projection task: Deduce outcome of sequence of actions
- Planning task: Find sequence of actions that achieves desired effect

- Examples:
  - At(Agent, [1,1], S_0) $\wedge$ At(G_1, [1,2], S_0) $\wedge$ ¬Holding(G_1, S_0)
  - Gold(G_1) $\wedge$ Adjacent([1,1], [1,2]) $\wedge$ Adjacent([1,2], [1,1])

Situation Calculus

- Projection / prediction / verification:
  - At(G_1, [1,1], Result([Go([1,1],[1,2]), Grab(G_1), Go([1,2],[1,1])], S_0))

- Planning:
  - ∃ seq At(G_1, [1,1], Result(seq, S_0))

Actions in Situation Calculus

- Possibility axioms:
  - At(Agent, x, s) $\wedge$ Adjacent(x,y) $\Rightarrow$ Poss(Go(x,y),s)
  - Gold(g) $\wedge$ At(Agent,x,s) $\wedge$ At(g, x, s) $\Rightarrow$ Poss(Grab(g),s)
- Effect axioms:
  - Poss(Go(x,y),s) $\Rightarrow$ At(Agent, y, Result(Go(x,y),S))
  - Poss(Grab(g),s) $\Rightarrow$ Holding(g, Result(Grab(g),S))
  - Poss(Release(g),s) $\Rightarrow$ ¬ Holding(g, Result(Release(g),S))
- Can prove now:
  - At(Agent, [1,2], Result(Go([1,1],[1,2]), S_0))
  - Can't show: At(G_1, [1,2], Result(Go([1,1],[1,2]), S_0))

Frame Problem

- How to handle the things that are NOT changed by an action?
- A actions, E effects per action, F fluents
- Representational frame problem: Size of knowledge base should depend on number of actions and effects, not fluents: O(AE)
- Inferential frame problem: Updates / prediction of t steps in O(Et) time

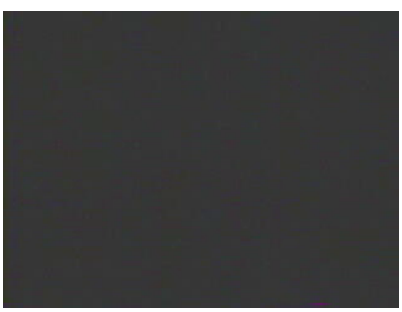
Representational Frame Problem

- Naïve solution O(AF):
  - At(o,x,s) ∧ (o≠ Agent) ∧ ¬Holding(o,s) ⇒ At(o,x,Result(Go(y,z),s))
- Successor-state axioms (Ray Reiter, '91) O(AE,F):
  - Action possible ⇒
    (fluent true in result state ⇔ Action's effect made it true
    v It was true before and action didn't change it)
  - Poss(a,s) ⇒
    (At(Agent,y,Result(a,s)) ⇔ a = Go(x,y)
    v (At(Agent,y,s) ∧ a ≠Go(y,z)))
  - Poss(a,s) ⇒
    F_i(Result(a,s)) ⇔ (a = A_1v a = A_2v …)
    v F_i(s) ∧ a ≠A_3∧ a ≠A_4…)

GOLOG

- Cognitive robotics
- Robot programming language based on Situation Calculus
- Extensions can handle concurrent actions, stochastic environments, and sensing
- Still too inefficient due to generality

GOLOG Application



Planning

- Given
  a logical description of the **initial situation**,
  a logical description of the **goal conditions**, and
  a logical description of a set of **possible actions**,
- find
  a **sequence of actions** (a **plan of action**) that brings us from the initial situation to a situation in which the goal conditions hold.

Input Representation

- *Description of initial state of world*
  E.g., Set of propositions:
  ((block a) (block b) (block c) (on-table a) (on-table b) (clear a) (clear b) (clear c) (arm-empty))
- *Description of goal: i.e. set of worlds*
  E.g., Logical conjunction
  Any world satisfying conjunction is a goal
  (and (on a b) (on b c)))
- *Description of available actions*

Planning vs. Problem-Solving

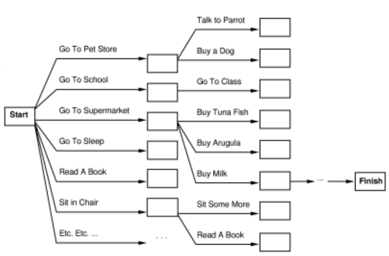
Basic difference: **Explicit, logic-based representation**

- States/Situations: descriptions of the world by logical formulae vs. data structures
  → agent can explicitly reason about and communicate with the world.
- Goal conditions as logical formulae vs. goal test (black box)
  → agent can reflect on its goals.
- Operators: Axioms or transformation on formulae vs. modification of data structures by programs
  → agent can gain information about the effects of actions by inspecting the operators.

Searching in State Space

We could search through the state space and thereby reduce planning to searching.

We can search forwards (**progression planning**):



Or alternatively, we can start at the goal and work backwards (**regression planning**).

Possible since the operators provide enough information

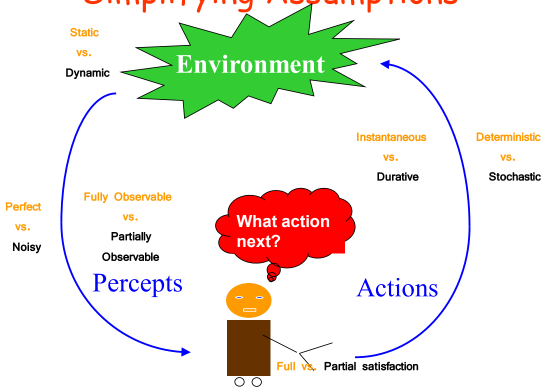
Ways to make "plans"

Generative Planning
Reason from first principles (knowledge of actions)
Requires formal model of actions

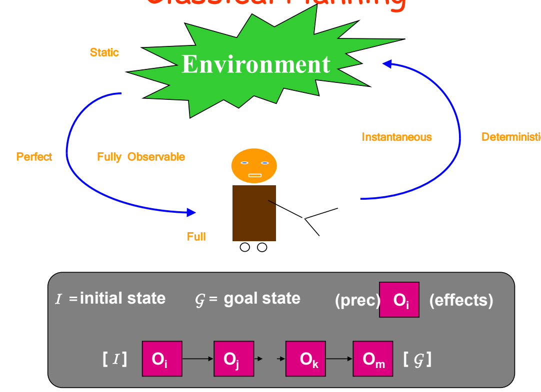
Case-Based Planning
Retrieve old plan which worked on similar problem
Revise retrieved plan for this problem

Reinforcement Learning
Act "randomly" - noticing effects
Learn reward, action models, policy

Simplifying Assumptions



Classical Planning

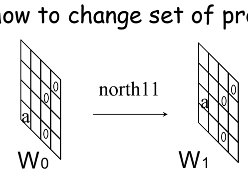


How Represent Actions?

- *Simplifying assumptions*
  - Atomic time
  - Agent is omniscient (no sensing necessary).
  - Agent is sole cause of change
  - Actions have deterministic effects
- *STRIPS representation*
  - World = set of true propositions (conjunction)
  - Actions:
    - Precondition: (conjunction of positive literals, ground, no functions)
    - Effects (conjunction of literals, ground, no function)
  - Goals = conjunctions (Rich ^ Famous)

STRIPS Actions

- Action = function: worldState→ worldState
- Precondition
  - says where function defined
- Effects
  - say how to change set of propositions



north11
precond: (and (agent-at 1 1)
(agent-facing north))

effect: (and (agent-at 1 2)
(not (agent-at 1 1)))

Action Schemata

- Instead of defining:
  pickup-A and **pickup-B** and …
- Define a schema:

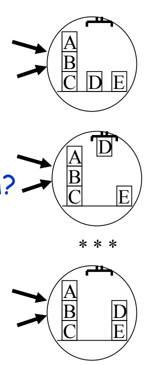
```
(:operator pick-up
    :parameters ((block ?ob1))
    :precondition (and (clear ?ob1)
                       (on-table ?ob1)
                       (arm-empty))
    :effect (and (not (clear ?ob1))
                 (not (on-table ?ob1))
                 (not (arm-empty))
                 (holding ?ob1)))
```


Forward State-Space Search

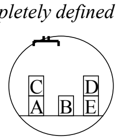
- Progression planning
- Initial state: set of positive ground literals (CWA: literals not appearing are false)
- Actions:
  - applicable if preconditions satisfied
  - add positive effect literals
  - remove negative effect literals
- Goal test: checks whether state satisfies goal
- Step cost: typically 1

Backward State-Space Search

- Regression planning
- Problem: Need to find predecessors of state
- Problem: Many possible goal states are equally acceptable.
- From which one does one search?

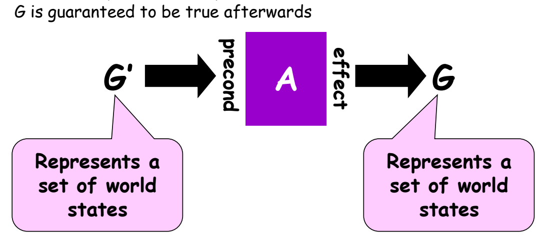


Initial State is completely defined



Regression

- Let G be a KR sentence (e.g. in logic)
- Relevance: needs to achieve one subgoal
- Consistency: does not undo any other subgoal
- Regressing a goal, *G*, thru an action, *A* yields the weakest precondition *G'*
  - Such that: if *G'* is true before *A* is executed *G* is guaranteed to be true afterwards



Regression Example

(and (clear C)
(on-table C)
(arm-empty)
(on A B))

(and (holding C)
(on A B))

pick-up :*parameters* ((block ?ob1))
:*precondition* (and (clear ?ob1)
(on-table ?ob1)
(arm-empty))
:*effect* (and (not (clear ?ob1))
(not (on-table ?ob1))
(not (arm-empty))
(holding ?ob1)))

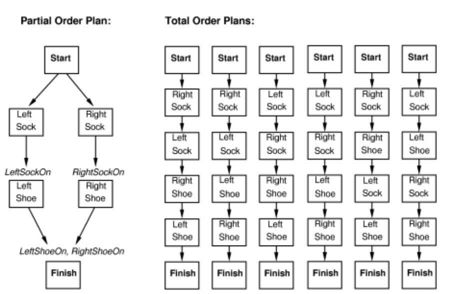
Remove positive effects
Add preconditions for A

Heuristics for State-Space Search

- Subgoal independence assumption:
  Cost of solving conjunction is sum of cost of solving each subgoal independently
  Optimistic: ignores negative interactions
  Pessimistic: ignores redundancy
- Relaxed problems:
  Remove all preconditions from actions and assume subgoal independence → heuristic is number of unsatisfied goals
  Remove preconditions and negative effects:
  - Goal(A^B^C)
  - Action(X,Effect:A^P)
  - Action(Y,Effect:B^C^Q)
  - Action(Z,Effect:B^P^Q)

  Set cover problem: NP-hard

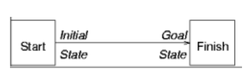
Plan = Sequence of Actions?



Searching in Plan Space

Instead of searching in state space, can search in space of all plans.

Initial state is **partial plan** containing only start and goal states:



Goal state is a complete plan that solves the given problem:

Representation of Partial Order (Non-Linear) Plans

During search, **plan** is represented by sets of

- actions (empty plan is Start and Finish only)
- ordering constraints (A<B: A before B)
- causal links $A_i \xrightarrow{c} A_j$ means "A_i produces the precondition c for A_j"
- open preconditions (not yet achieved preconditions)
- variable assignments x = t, where x is a variable and t is a constant or a variable.

- Solutions to planning problems must be **complete** and **consistent**.

Completeness and Consistency

Complete: Every precondition of every step is fulfilled

Consistent: No cycles in ordering constraints and no conflicts with causal links

Shoe example solution:

Actions: { RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish}

Orderings: { RightSock < RightShoe, LeftSock < LeftShoe}

Links: { RightSock $\xrightarrow{RightSockOn}$ RightShoe, LeftSock $\xrightarrow{LeftSockOn}$ LeftShoe,

RightShoe $\xrightarrow{RightShoeOn}$ Finish, LeftShoe $\xrightarrow{LeftShoeOn}$ Finish}

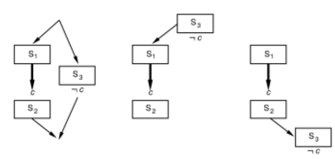
OpenPreconditions: {}

Searching in Plan Space

- Successor function: (plan refinement)
  - pick open precondition p and check all actions that generate p
  - consistency:
    - add causal link and ordering constraint(s)
    - check whether there are potential conflicts (clobberers) and try to protect violated links

- Goal test: No open preconditions

Protection of Causal Links



(a) Conflict: S_3 threatens the causal link between S_1 and S_2.

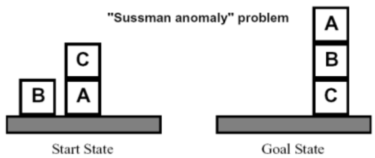
Conflict solutions:

(b) **Demotion**: Place threatening step before causal link

(c) **Promotion**: Place threatening step after causal link

Blocks World Example

"Sussman anomaly" problem



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

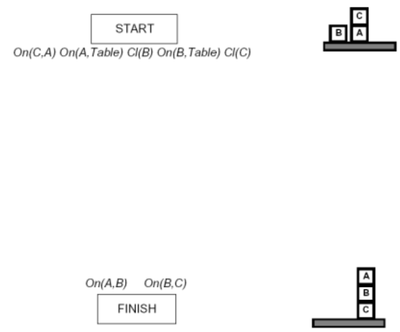
Clear(x) On(x,z)

PutOnTable(x)

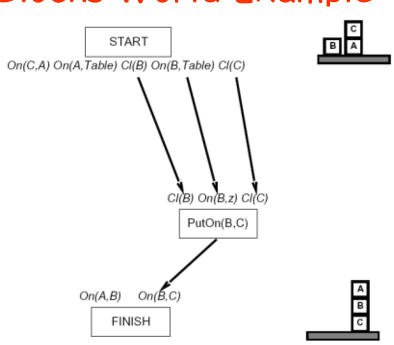
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

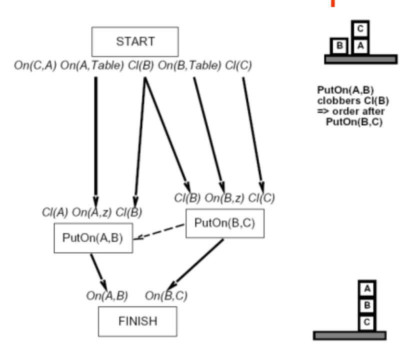
Blocks World Example



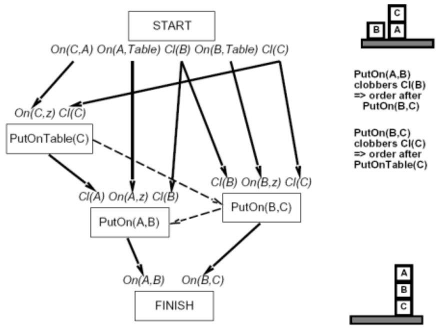
Blocks World Example



Blocks World Example



Blocks World Example



POP Algorithm

Correctness: Every result of the POP algorithm is a complete, correct plan.

Completeness: If breadth-first-search is used, the algorithm finds a solution, given one exists.

GraphPlan: Basic idea

- Construct a graph that encodes constraints on possible plans
- Use this "planning graph" to constrain search for a valid plan:
  If valid plan exists, it's a subgraph of the planning graph
- Planning graph can be built for each problem in polynomial time

Problems handled by GraphPlan*

- Pure STRIPS operators:
  conjunctive preconditions
  no negated preconditions
  no conditional effects
  no universal effects
- Finds "shortest parallel plan"
- Sound, complete and will terminate with failure if there is no plan.

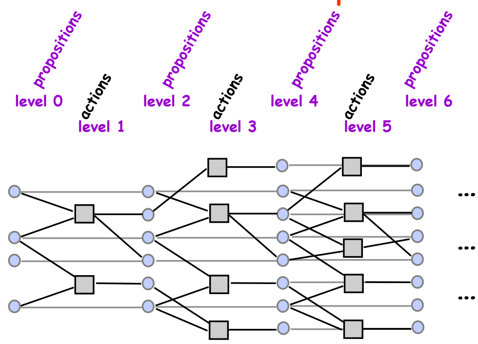
*Version in [Blum& Furst IJCAI 95, AIJ 97], later extended to handle all these restrictions [Koehler et al 97]

Graphplan

- Phase 1 - Graph Expansion
  Necessary (insufficient) conditions for plan existence
  Local consistency of plan-as-CSP
- Phase 2 - Solution Extraction
  Variables
  - action execution at a time point

  Constraints
  - goals, subgoals achieved
  - no side-effects between actions

The Plan Graph



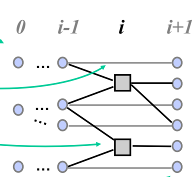
Note: a few noops missing for clarity

Graph Expansion

Proposition level 0
- initial conditions

Action level i
- no-op for each proposition at level i-1
- action for each operator instance whose preconditions exist at level i-1

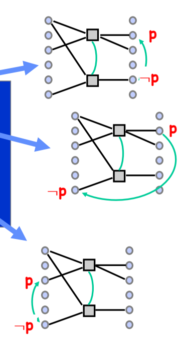
Proposition level i
- effects of each no-op and action at level i

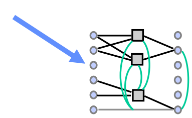


Mutual Exclusion

Two actions are mutex if
- one clobbers the other's effects or preconditions
- they have mutex preconditions

Two proposition are mutex if
- one is the negation of the other
- all ways of achieving them are mutex





Graphplan

- Create level 0 in planning graph
- Loop

  If goal ⊆ contents of highest level (nonmutex)

  Then search graph for solution

  - If find a solution then return and terminate

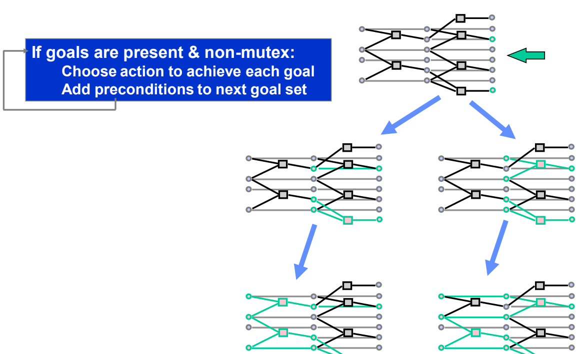
  Else extend graph one more level

A kind of double search: forward direction checks necessary (but insufficient) conditions for a solution, ... Backward search verifies...

Searching for a Solution Plan

- Backward chain on the planning graph
- Achieve goals level by level
- At level k, pick a subset of non-mutex actions to achieve current goals. Their preconditions become the goals for k-1 level.
- Build goal subset by picking each goal and choosing an action to add. Use one already selected if possible. Do forward checking on remaining goals (backtrack if can't pick non-mutex action)

Searching for a Solution



Dinner Date

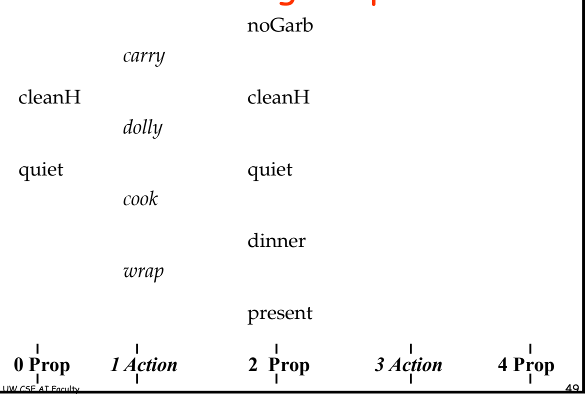
Initial Conditions: (:and (cleanHands) (quiet))

Goal: (:and (noGarbage) (dinner) (present))

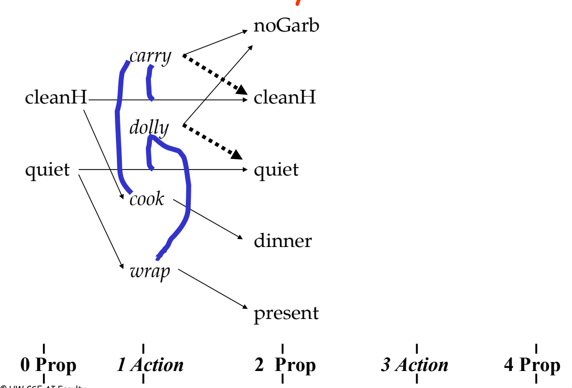
Actions:

(:operator **carry** *:precondition*
:effect (:and (noGarbage) (:not (cleanHands)))
(:operator **dolly** *:precondition*
:effect (:and (noGarbage) (:not (quiet)))
(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))
(:operator **wrap** *:precondition* (quiet)
:effect (present))

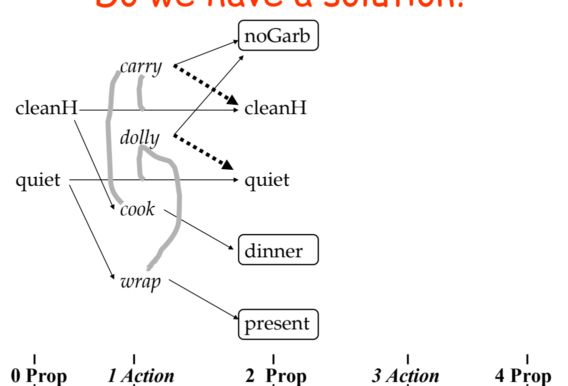
Planning Graph



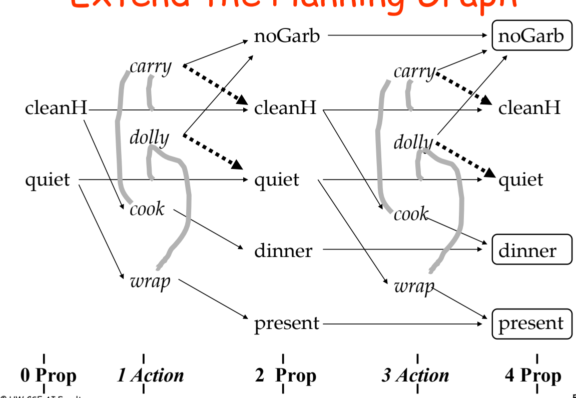
Are there any exclusions?



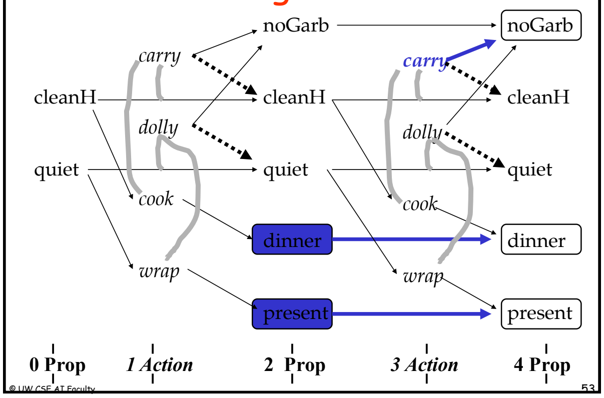
Do we have a solution?



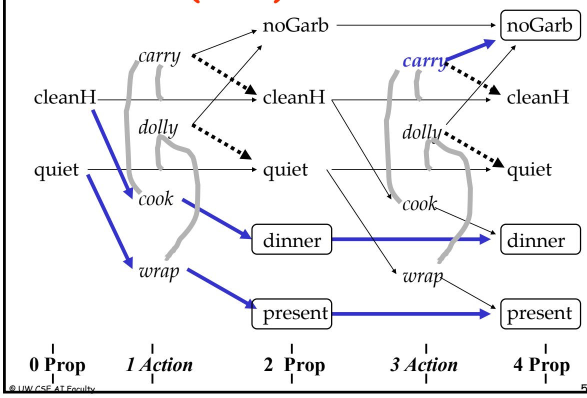
Extend the Planning Graph



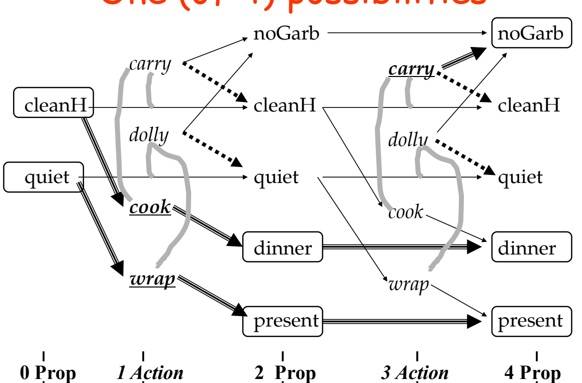
Searching Backwards



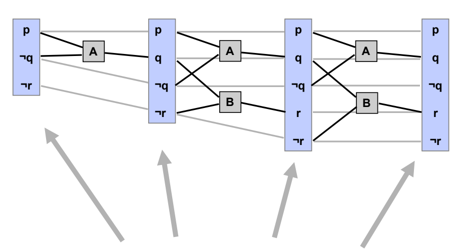
One (of 4) Possibilities



One (of 4) possibilities

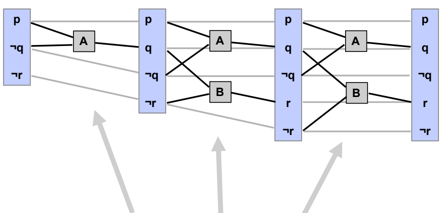


Observation 1



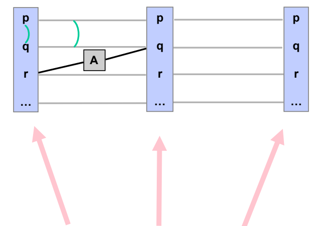
Propositions monotonically increase
(always carried forward by no-ops)

Observation 2



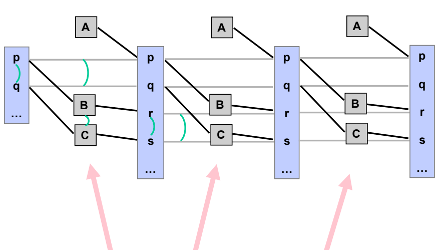
Actions monotonically increase

Observation 3



Proposition mutex relationships monotonically decrease

Observation 4



Action mutex relationships monotonically decrease

Observation 5

Planning Graph 'levels off'.

- After some time k all levels are identical
- Because it's a finite space, the set of literals never decreases and mutexes don't reappear.

The Last Word on Planning: SATPlan

- Idea: test the *satisfiability* of the logical sentence:
  (initial state) ∧ (all possible action descriptions for t steps) ∧ (goal achieved at step t)
- Create and test sentence for each t, t = 0, 1, 2, …, T_{max}
- Action descriptions include
  1. Successor-state axioms from situation calculus (superscript denotes t)
     E.g., $At(P1,JFK)^1 \Leftrightarrow (At(P1,SFO)^0 \wedge Fly(P1,SFO,JFK)^0) \vee (At(P1,JFK)^0 \wedge \neg Fly(P1,JFK,SFO)^0)$
  2. Precondition axioms E.g., $Fly(P1,SFO,JFK)^0 \Rightarrow At(P1,SFO)^0$
  3. State constraints.
     E.g., $\forall p, x, y, t \quad (x \neq y) \Rightarrow \neg(At(p,x)^t \wedge At(p,y)^t)$

Planning using SATPlan

- Sentence to be tested (for a particular t):
  (initial state) ∧ (all possible action descriptions) ∧ (goal)
- A model will assign true to actions that are part of correct plan and false to other actions
  - If no plan exists, sentence will be unsatisfiable
- Use SAT solver such as DPLL or WalkSAT to test satisfiability (and find plan if one exists)
- SATPlan can handle large planning problems
  - E.g., Up to 30-step plans in blocks world

Some Applications of Planning

- Assembly line planning at Hitachi
- Software procurement planning at Price Waterhouse
- Back-axle assembly planning at Jaguar Cars
- Logistics planning in the US Navy
- Scheduling mission-command sequences for satellites
- Observation planning for Hubble telescope
- Spacecraft control for Deep Space One probe
- Etc.

Planning Summary

- Problem solving algorithms that operate on explicit propositional representations of states and actions.
- Make use of specific heuristics.
- STRIPS: restrictive propositional language
- State-space search: forward (progression) / backward (regression) search
- Partial order planners search space of plans from goal to start, adding actions to achieve goals
- GraphPlan: Generates planning graph to guide backwards search for plan
- SATplan: Converts planning problem into propositional axioms. Uses SAT solver to find plan.