CSE 473

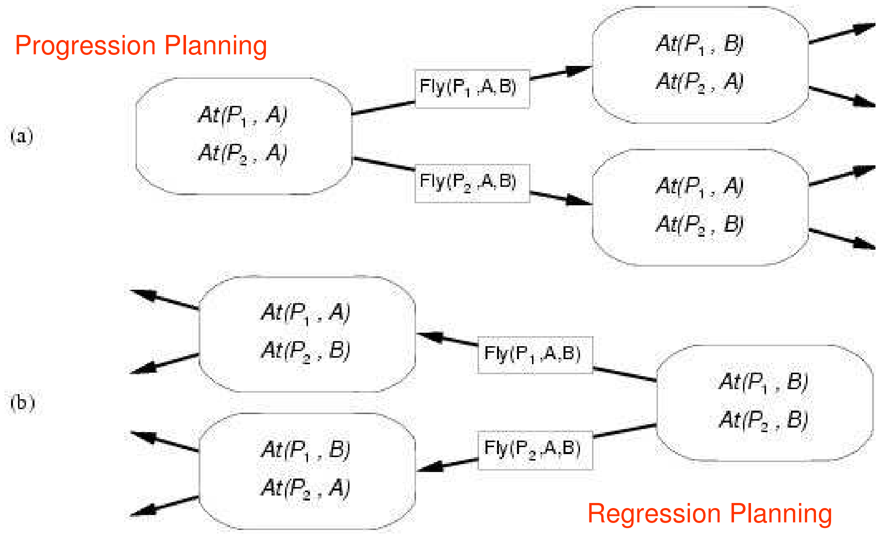
Chapter 11

Planning, Planning, and more Planning

Things we will munch on today

- Regression Planning
- Partial order Planning
- GRAPHPlan

Regressing to Last Lecture



Regression Planning

- How to determine predecessors?
  What are the states from which applying a given action leads to the goal?
  Goal state = *At(C1, B)* ∧ *At(C2, B)* ∧ … ∧ *At(C20, B)*
  Relevant action for first conjunct: *Unload(C1,p,B)*
  Works only if pre-conditions are satisfied.
  Previous state= *In(C1, p)* ∧ *At(p, B)* ∧ *At(C2, B)* ∧ …
  ∧ *At(C20, B)*
- Actions must not undo desired literals (consistent)
- Main advantage: only relevant actions are considered
  Often much lower branching factor than forward search.
- Admissible heuristics can be used with A* search to find optimal solutions (see Section 11.2)

Partial-order planning

- Progression and regression planning are *totally ordered plan search* methods.
  Can't work on subproblems independently and combine solutions
- Partial-order planning uses a "least commitment strategy":
  Find *subplans* for achieving subgoals and combine them to get final plan

Shoe example

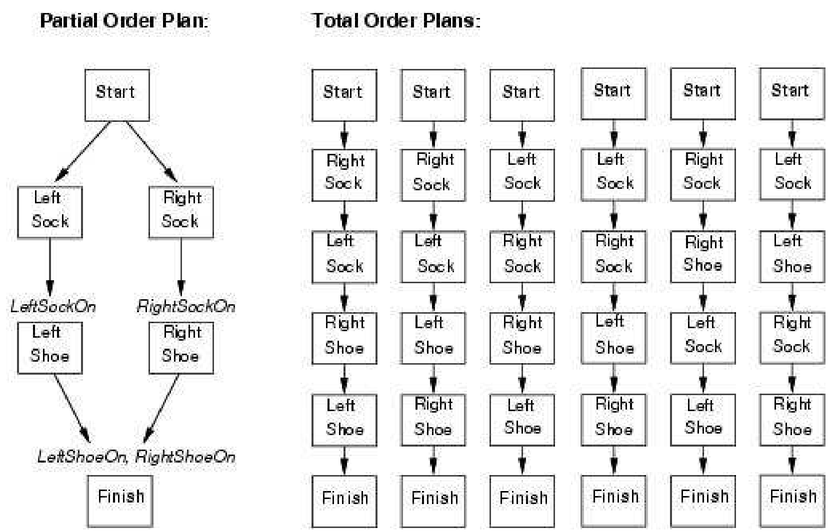
Init()

Goal(RightShoeOn $\wedge$ LeftShoeOn)

Action(RightShoe,
  PRECOND: RightSockOn
  EFFECT: RightShoeOn)
Action(RightSock,
  PRECOND:
  EFFECT: RightSockOn)
Action(LeftShoe,
  PRECOND: LeftSockOn
  EFFECT: LeftShoeOn)
Action(LeftSock,
  PRECOND:
  EFFECT: LeftSockOn)

Planner: Two subplans (actions can be interleaved)
  (1) leftsock, leftshoe and (2) rightsock, rightshoe

Partial-order planning (POP)



POP as a search problem

- States are (unfinished) plans.
  The empty plan contains only *Start* and *Finish* actions.
- Each plan has 4 components:
  A set of "actions" (steps of the plan)
  A set of ordering constraints: A < B (A before B)
    - Cycles represent contradictions.
  A set of causal links between actions
    - The plan may not be extended by adding a new action C that conflicts with the causal link
  A set of open preconditions.
    - Preconditions not achieved by actions in the plan
- A partial order plan can be executed by repeatedly choosing *any* of the possible next actions.
  This flexibility is a benefit in non-cooperative environments.

POP as a search problem

- Initial plan contains:
  - *Start* and *Finish*
  - ordering constraint *Start < Finish*
  - no causal links yet
  - all the preconditions in *Finish* are open (yet to be satisfied)
- Successor function :
  - picks one open precondition *p* on an action *B* and
  - generates a successor plan for every possible consistent way of choosing action *A* that achieves *p*.
- Test goal
- Heuristic function used to decide which open precondition to pick (e.g., most constrained first)

Blocks World Example

Actions with Preconditions and Effects:

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

PutOn(x,y)

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

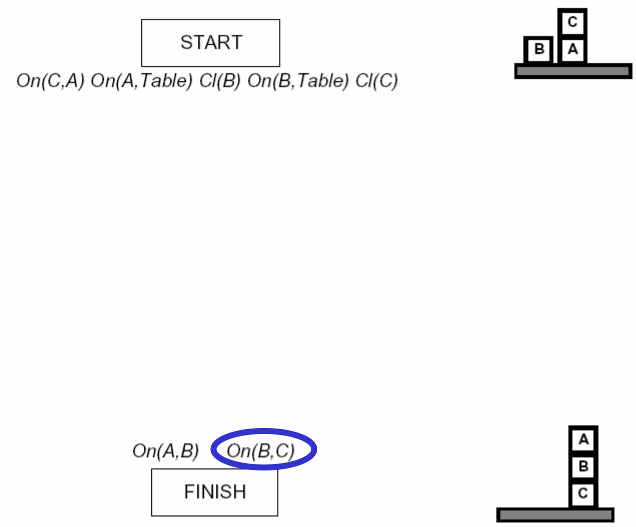
Clear(x) On(x,z)

PutOnTable(x)

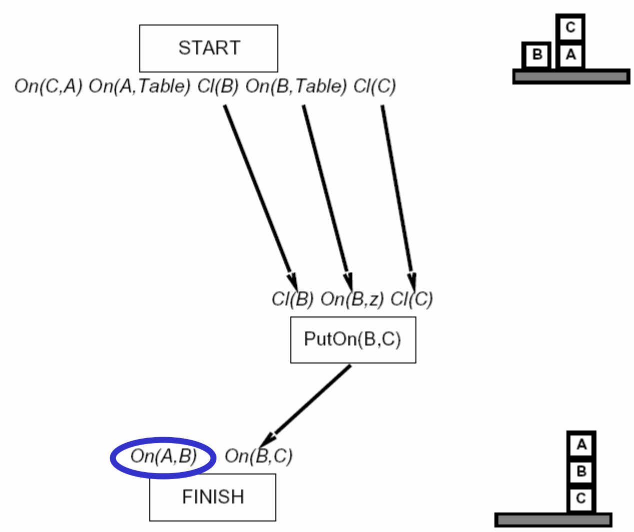
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

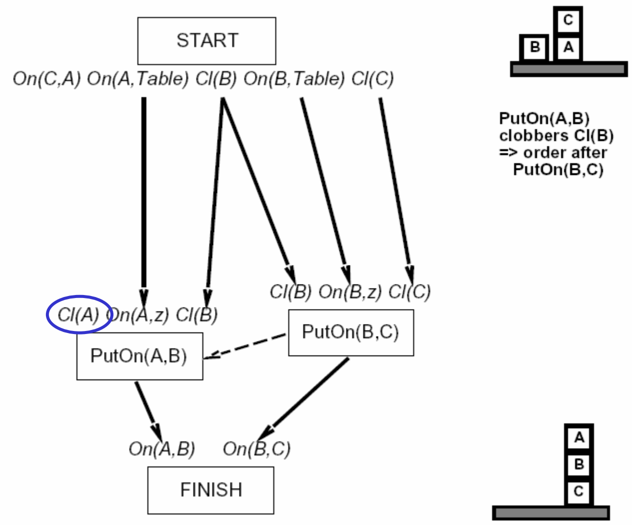
Blocks World Example



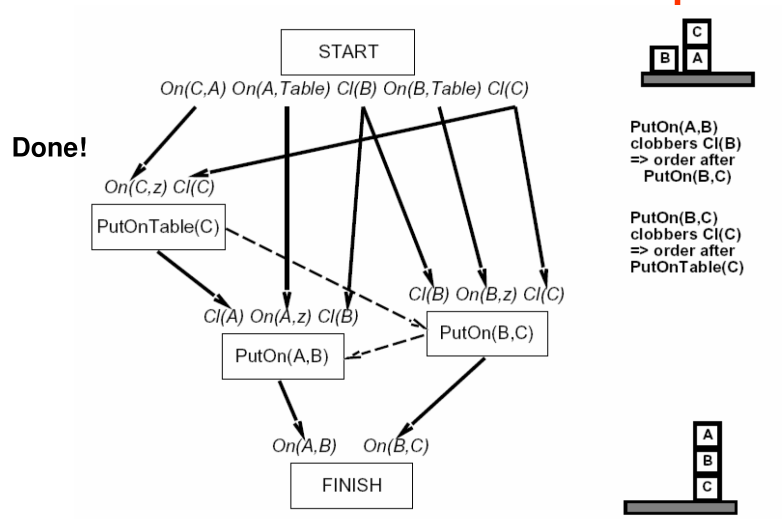
Blocks World Example



Blocks World Example



Blocks World Example



Final Plan: START, PutOnTable(C), PutOn(B,C), PutOn(A,B), FINISH

POP Algorithm

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

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

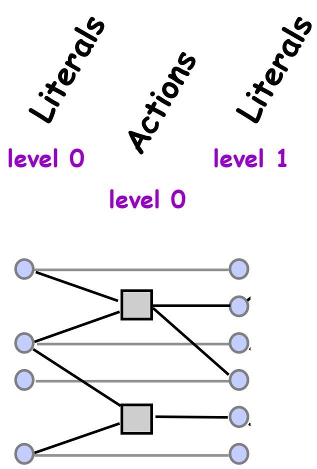
GraphPlan Algorithm: Basic idea

- 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 is a subgraph of the planning graph
- Planning graph can be built for each problem in polynomial time
- Sound, complete and will terminate with failure if there is no plan.

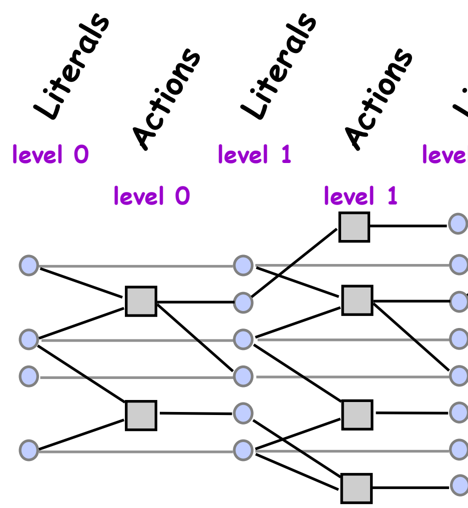
GraphPlan

- Phase 1 - Graph Expansion
  - Necessary (but not sufficient) conditions for plan existence
  - Constraints between actions/effects elucidated
- Phase 2 - Solution Extraction
  - Backwards search through graph to find actions satisfying goals

The Plan Graph



Expansion of Plan Graph

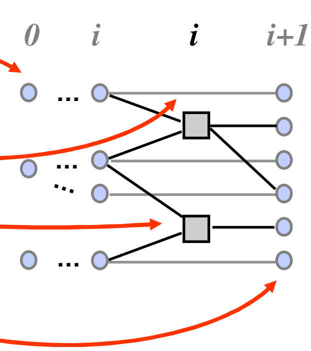


Graph Expansion

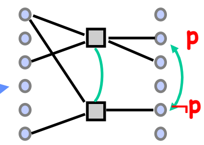
Initial State: level 0
- initial set of literals

Action level i
- no-op for each literal at level i
- action for each case where
  - preconditions exist at level i

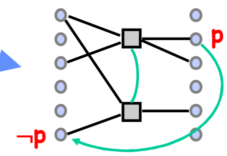
Literals level i+1
- effects of each action and no-op at level i+1



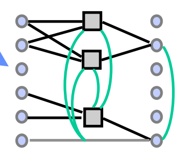
Mutual Exclusion

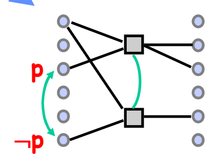


Two actions are mutex if
- one clobbers the other's effects or preconditions, or
- they have mutex preconditions



Two literals are mutex if
- one is the negation of the other, or
- all ways of achieving them are mutex





GraphPlan Algorithm

- Create level 0 in planning graph
- Loop

  If goals $\subseteq$ current level literals and all non-mutex
    then search graph for solution
  - If a solution was found, return and terminate

  Else extend graph one more level

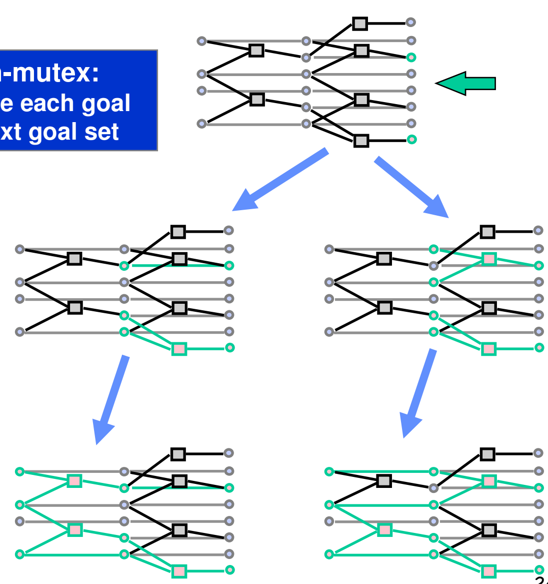
Forward direction checks necessary conditions for a solution...
Backward search constructs actual solution...

Searching for a Solution Plan

- Backwards search 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.
- At level 0, check to see if all goals satisfied

Searching for a Solution

If goals are present & non-mutex:
Choose action to achieve each goal
Add preconditions to next goal set



Planning a Dinner Date

Initial Conditions: cleanHands $\wedge$ quiet

Goal: noGarbage $\wedge$ dinner $\wedge$ gift

Actions:

carry *precondition:*
effect: noGarbage $\wedge$ ¬cleanHands

dolly *precondition:*
effect: noGarbage $\wedge$ ¬quiet

cook *precondition:* cleanHands
effect: dinner

wrap *precondition:* quiet
effect: gift

Planning Graph

noGarb

carry

cleanH cleanH

dolly

quiet quiet

cook

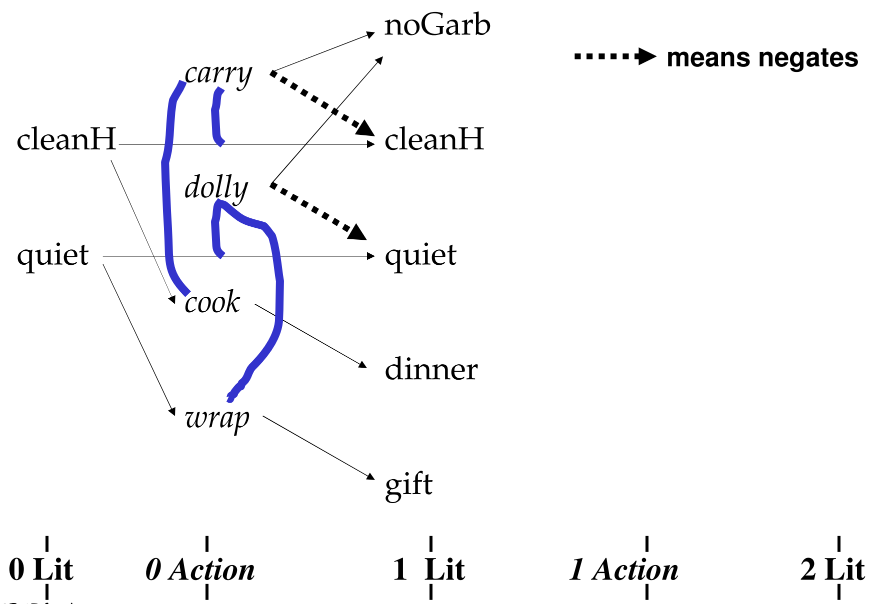
dinner

wrap

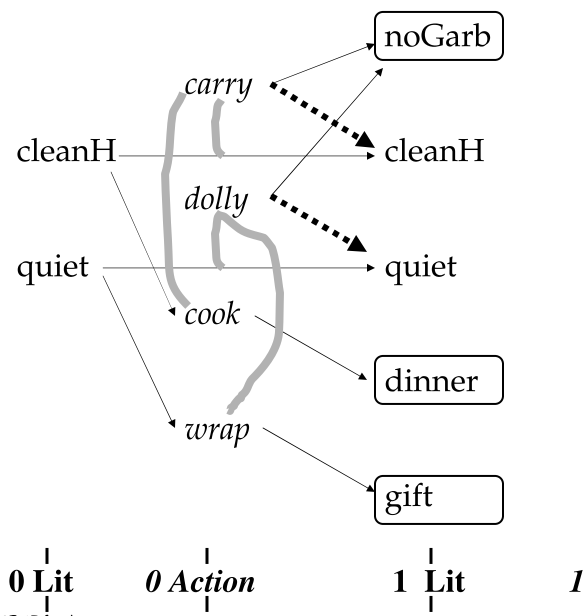
gift

0 Lit ***0 Action*** **1 Lit** ***1 Action*** **2 Lit**

Are there any mutexes?

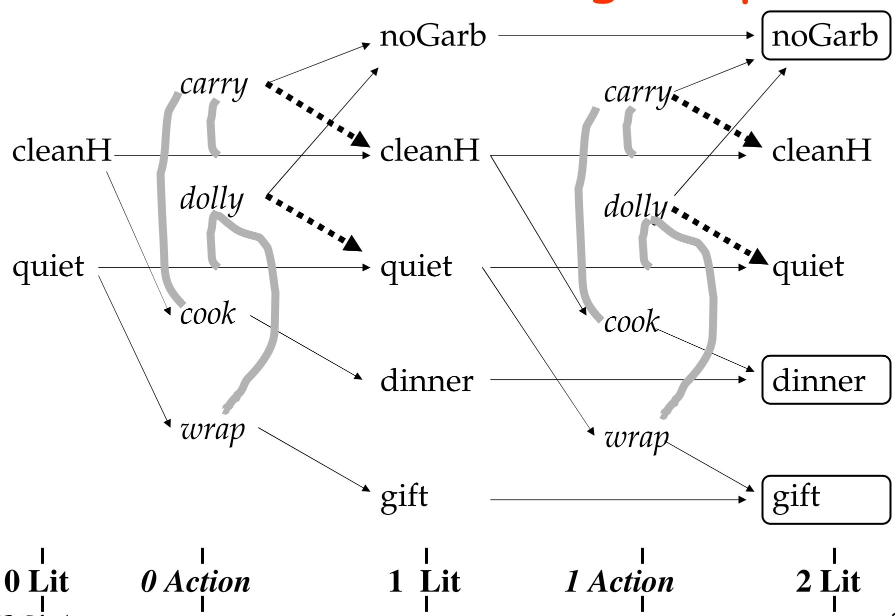


Do we have a solution?

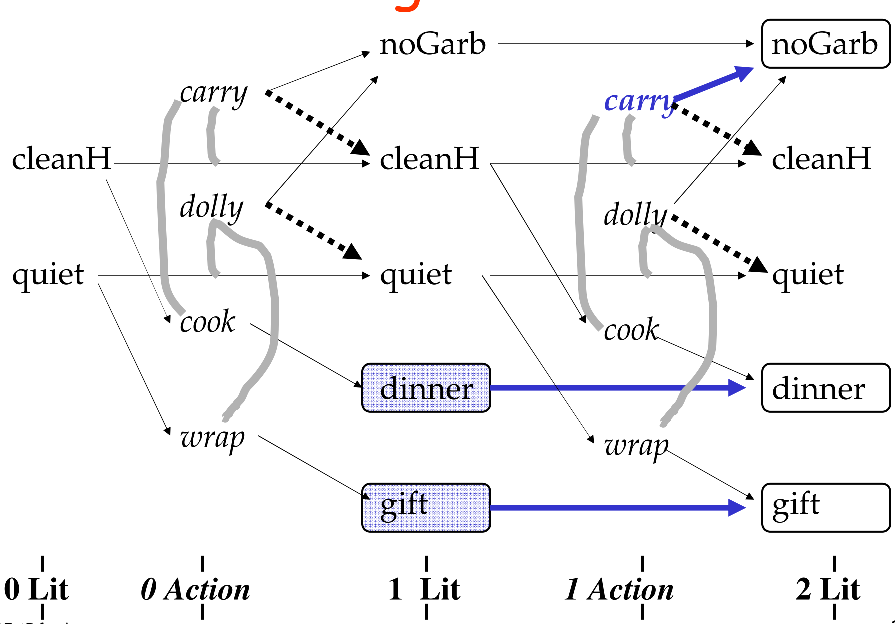


Any non-mutex ways of achieving all goals?

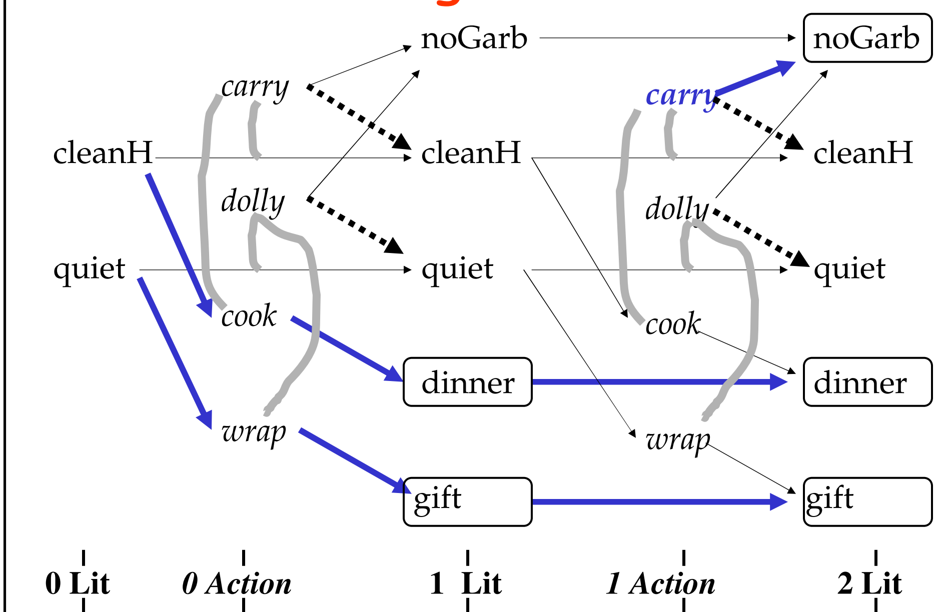
Extend the Planning Graph



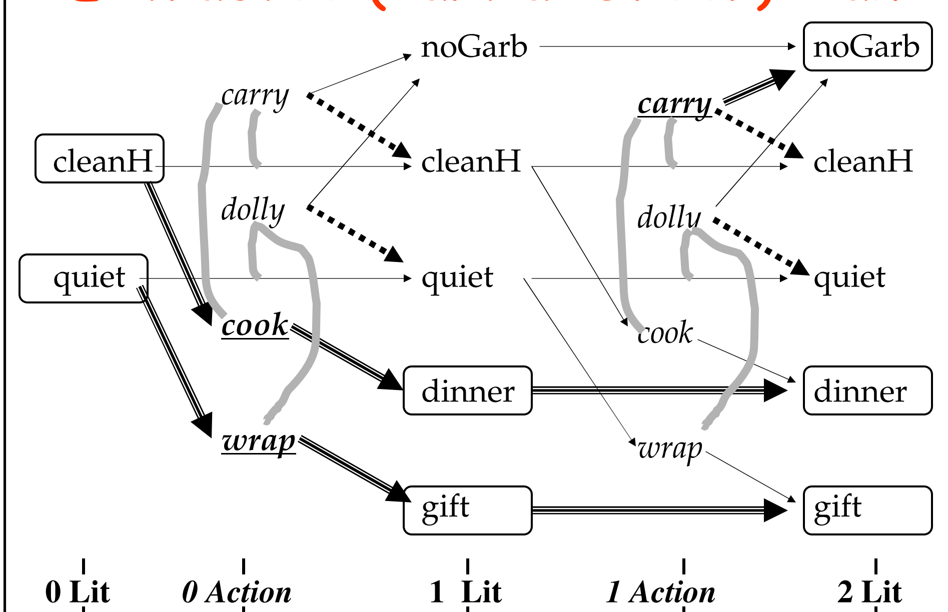
Searching Backwards



Searching Backwards



Extracted (Partial Order) Plan



Next Time

- SATPlan
- Uncertainty!
- Things to do:
  - Go over HW #4 (programming project)
  - Pick programming project partner(s)