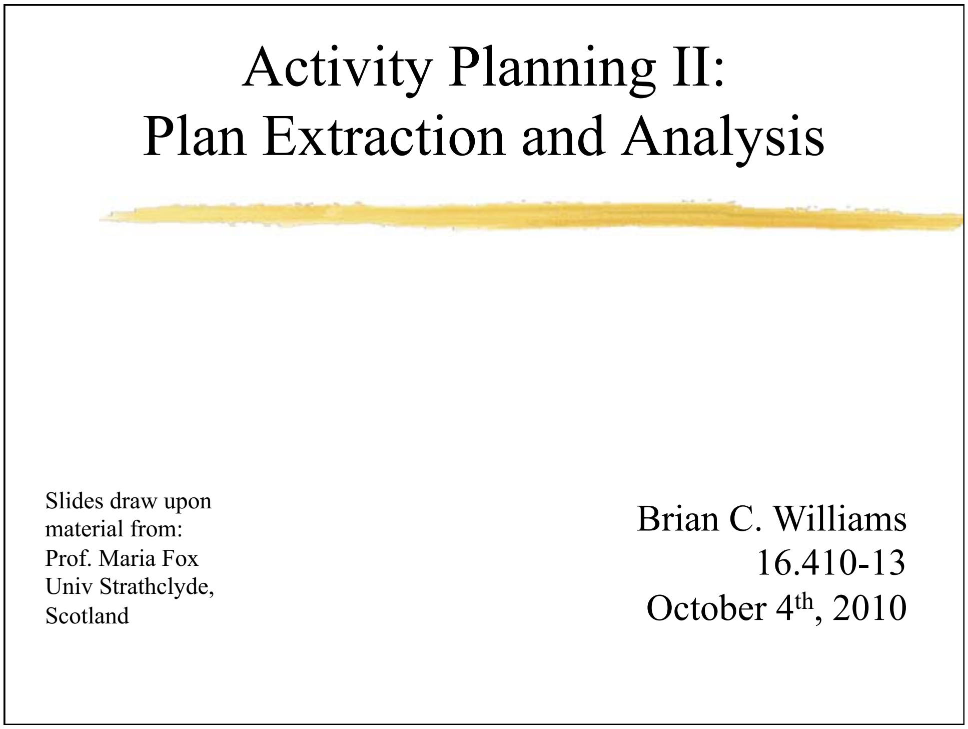
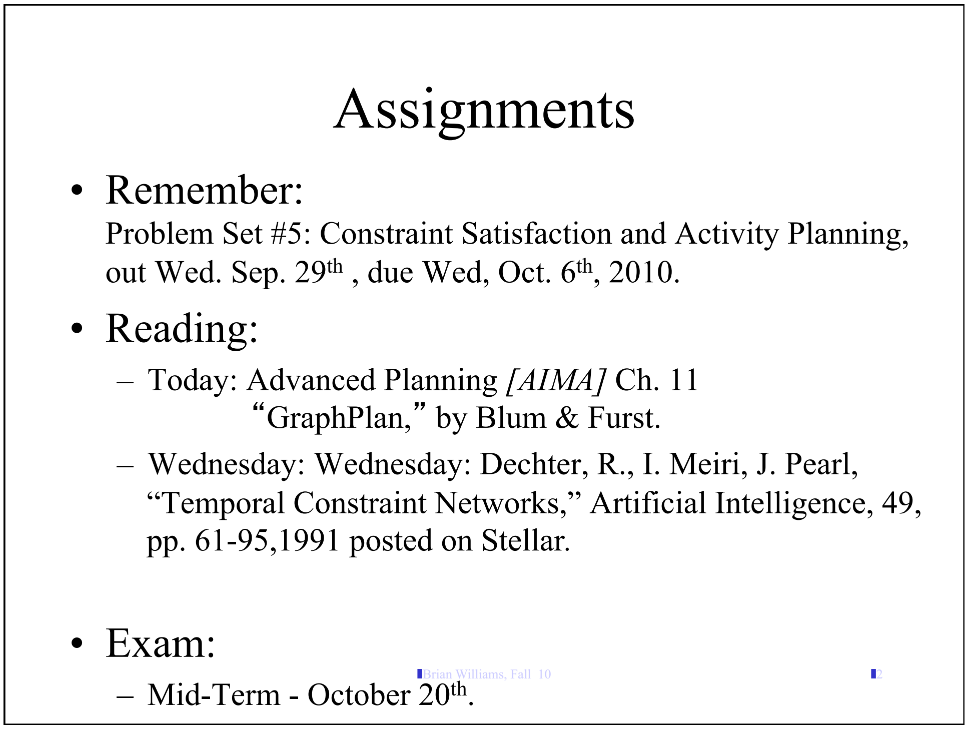


Activity Planning II:
Plan Extraction and Analysis

Slides draw upon
material from:
Prof. Maria Fox
Univ Strathclyde,
Scotland

Brian C. Williams
16.410-13
October 4th, 2010



Assignments

- Remember:
  Problem Set #5: Constraint Satisfaction and Activity Planning, out Wed. Sep. 29th , due Wed, Oct. 6th, 2010.
- Reading:
  - Today: Advanced Planning *[AIMA]* Ch. 11
    "GraphPlan," by Blum & Furst.
  - Wednesday: Wednesday: Dechter, R., I. Meiri, J. Pearl, "Temporal Constraint Networks," Artificial Intelligence, 49, pp. 61-95,1991 posted on Stellar.
- Exam:
  - Mid-Term - October 20th.

Example Problem: Dinner Date

Initial Conditions: (:init (cleanHands) (quiet))

Goal: (:goal (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))

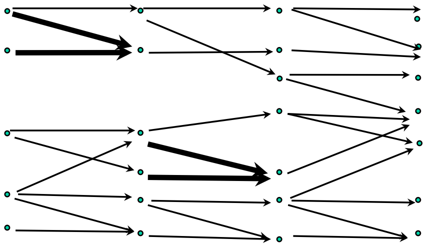
(:operator **wrap** *:precondition* (quiet)
:effect (present))

+ noops

Solution: (Cook, Wrap, Carry)

Approach: Graph Plan

1. Construct compact encoding of candidates that prunes many invalid plans – *Plan Graph.*
2. Generate plan by searching for a consistent subgraph that achieves the goals.



Proposition Init State | Action Time 1 | Proposition Time 1 | Action Time 2

Plan Graph

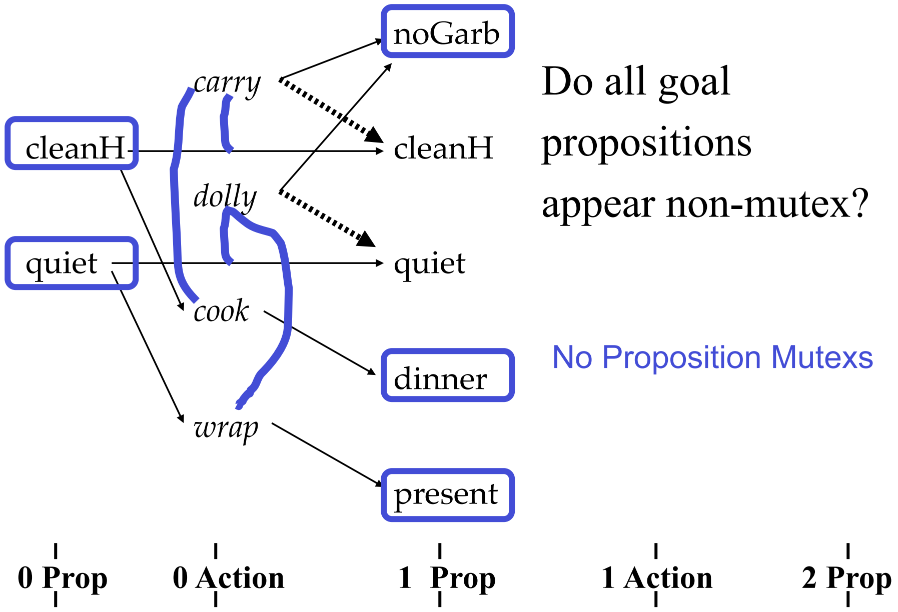
- Compactly encodes the space of consistent plans,
- while pruning . . .
  1. partial states and actions at each time i that are not reachable from the initial state.
  2. pairs of propositions and actions that are mutually inconsistent at time i.
  3. plans that cannot reach the goals.

Graph Plan

- Create plan graph level 1 from initial state
- Loop
  1. If goal $\subseteq$ propositions of the highest level (nonmutex),
  2. Then search graph for solution
     - If solution found, then return and terminate.
  3. Extend graph one more level.

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

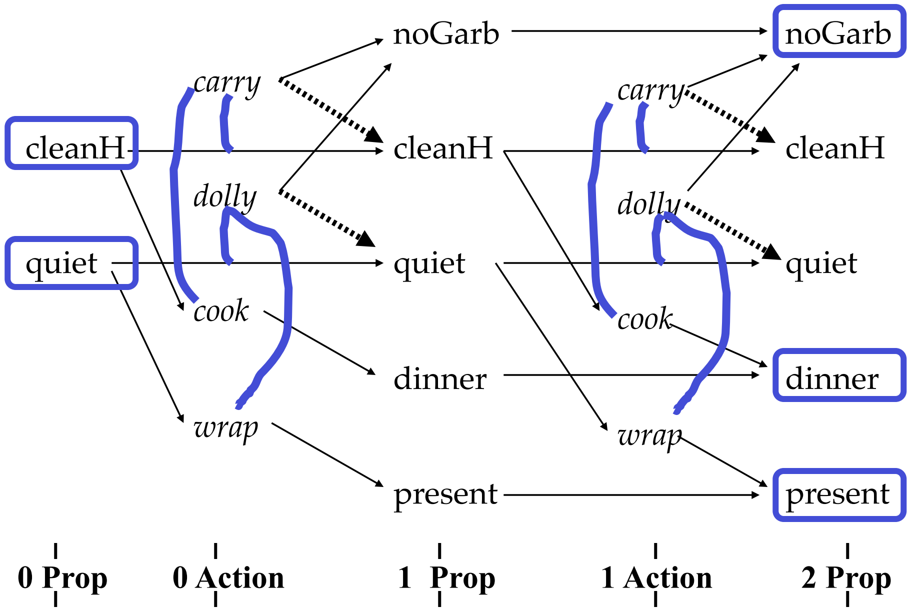
Layer 1: Add Proposition Mutexs



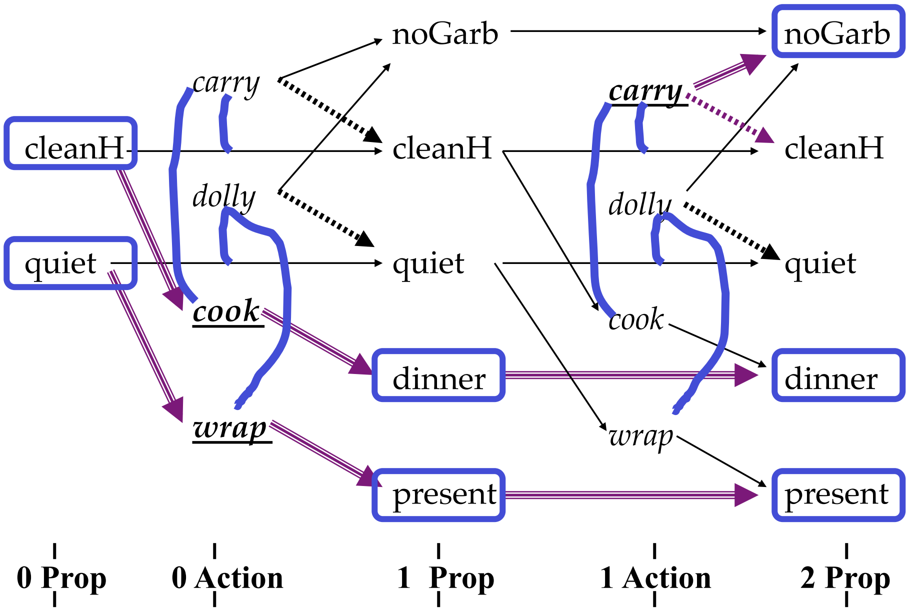
Do all goal propositions appear non-mutex?

No Proposition Mutexs

Round 2: Extend the Plan Graph

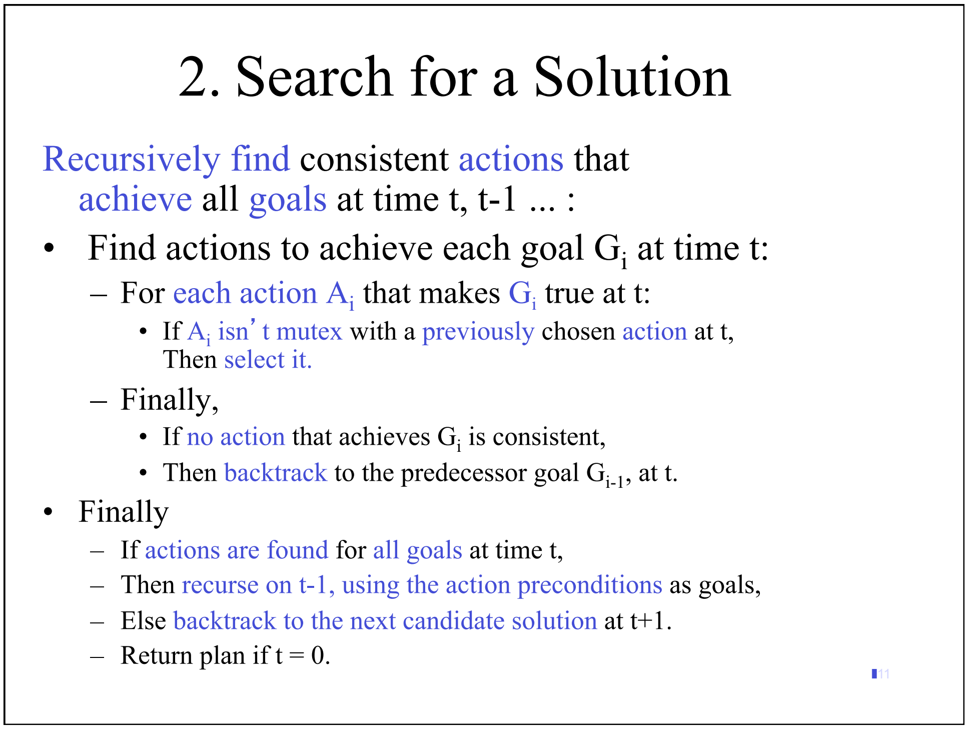


Search Graph for Solution



Outline

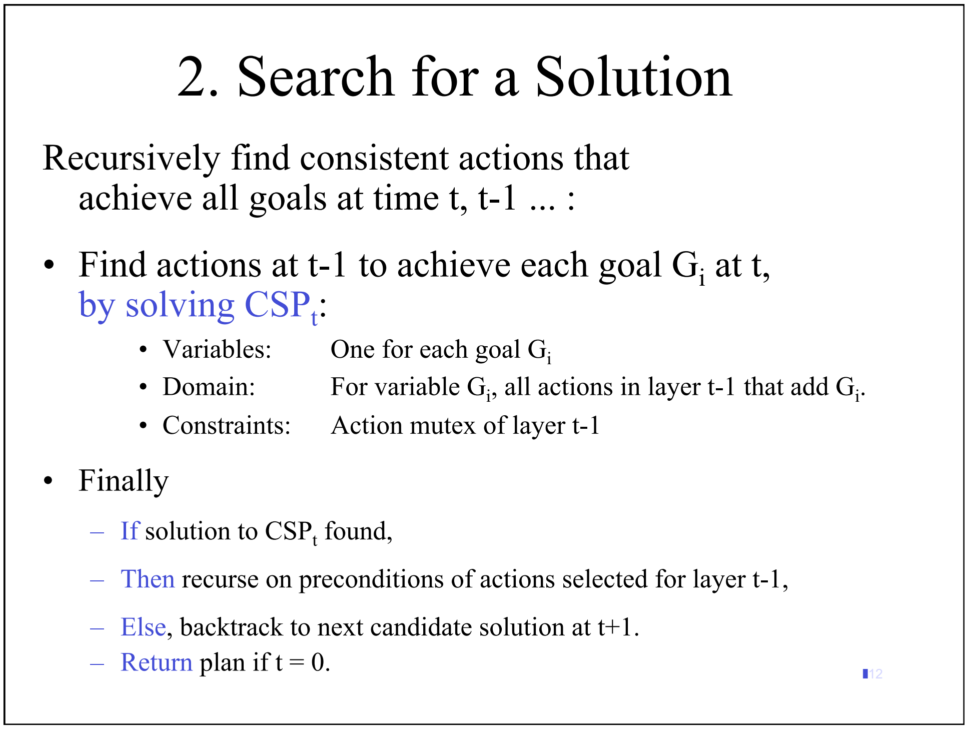
- Graph Plan
  - – Solution Extraction
  - – Memos
  - – Properties
  - – Termination with Failure



2. Search for a Solution

Recursively find consistent actions that achieve all goals at time t, t-1 ... :

- Find actions to achieve each goal G_i at time t:
  - For each action A_i that makes G_i true at t:
    - If A_i isn’ t mutex with a previously chosen action at t, Then select it.
  - Finally,
    - If no action that achieves G_i is consistent,
    - Then backtrack to the predecessor goal G_{i-1}, at t.
- Finally
  - If actions are found for all goals at time t,
  - Then recurse on t-1, using the action preconditions as goals,
  - Else backtrack to the next candidate solution at t+1.
  - Return plan if t = 0.



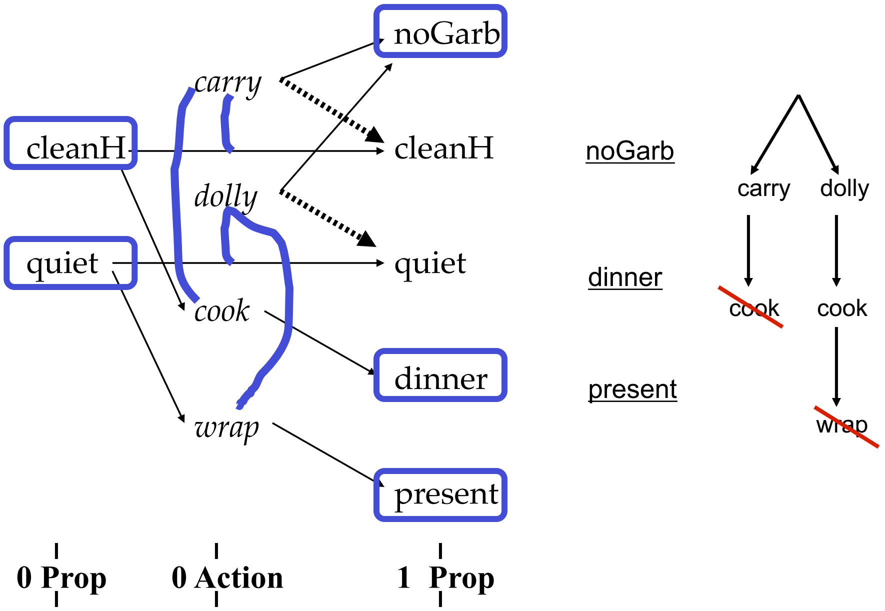
2. Search for a Solution

Recursively find consistent actions that achieve all goals at time t, t-1 ... :

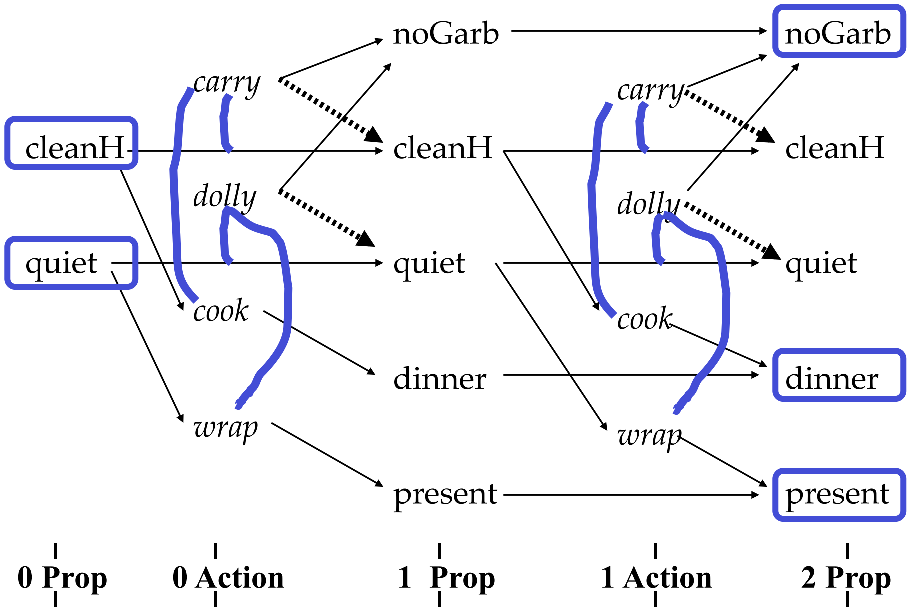
- Find actions at t-1 to achieve each goal G_i at t, by solving CSP_t:
  - Variables: One for each goal G_i
  - Domain: For variable G_i, all actions in layer t-1 that add G_i.
  - Constraints: Action mutex of layer t-1
- Finally
  - If solution to CSP_t found,
  - Then recurse on preconditions of actions selected for layer t-1,
  - Else, backtrack to next candidate solution at t+1.
  - Return plan if t = 0.

- Favor No-ops over other actions.
  - guarantees the plan will avoid redundant plan steps.

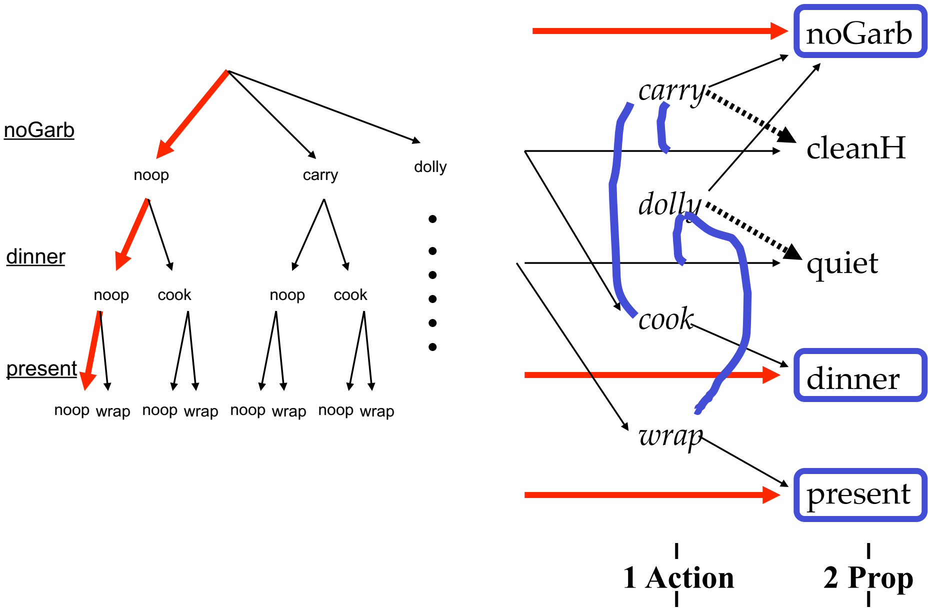
Search Action Layer 0



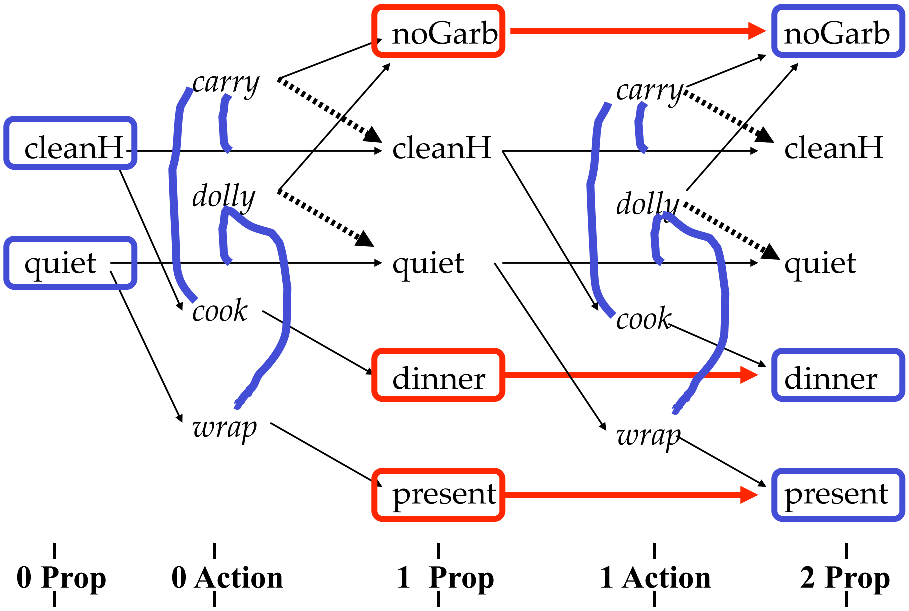
Extend & Search Action Layer 1



Search Action Layer 1



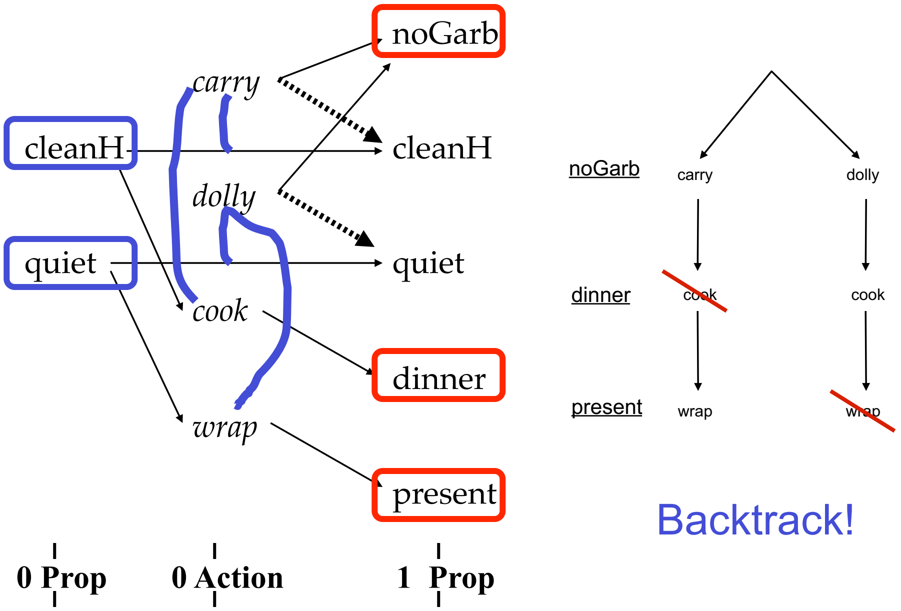
Search Action Layer 0



noGarb
carry
cleanH
dolly
quiet
cook
dinner
wrap
present
carry
cleanH
dolly
quiet
cook
wrap
noGarb
cleanH
quiet
dinner
present
noGarb
cleanH
quiet
dinner
present

0 Prop
0 Action
1 Prop
1 Action
2 Prop

Search Action Layer 0

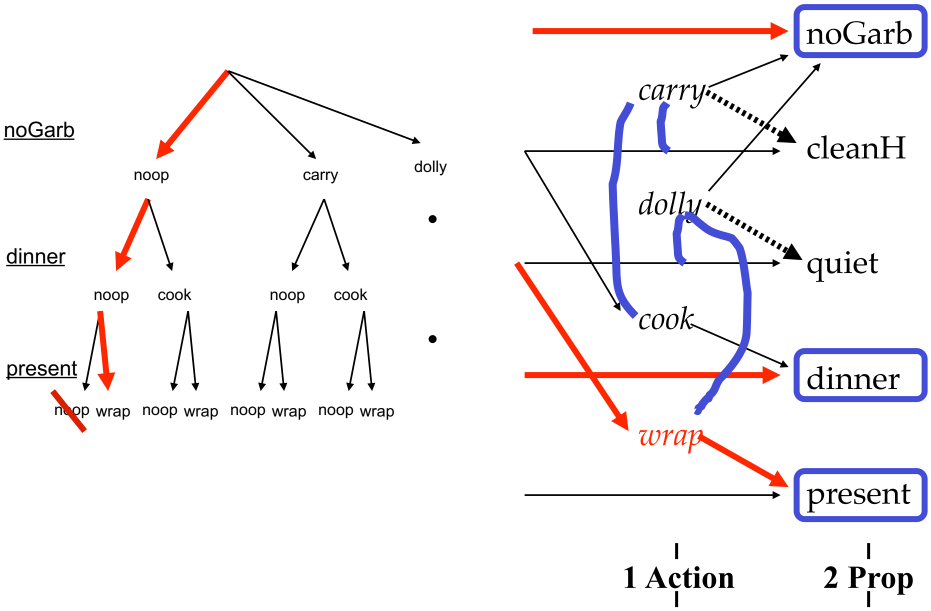


noGarb
carry
cleanH
dolly
quiet
cook
dinner
wrap
present
cleanH
quiet
noGarb
dinner
present
carry
dolly
~~cook~~
cook
wrap
~~wrap~~

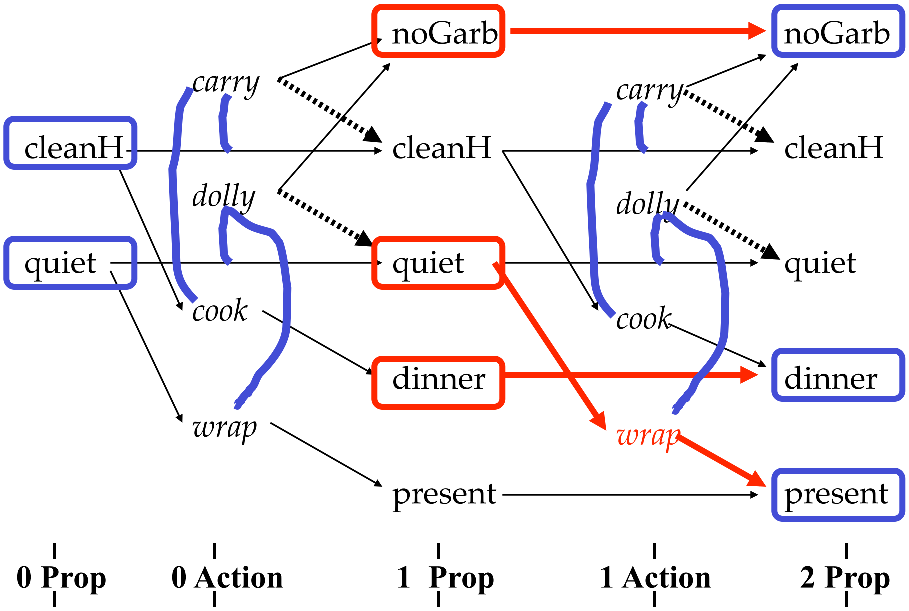
Backtrack!

0 Prop
0 Action
1 Prop

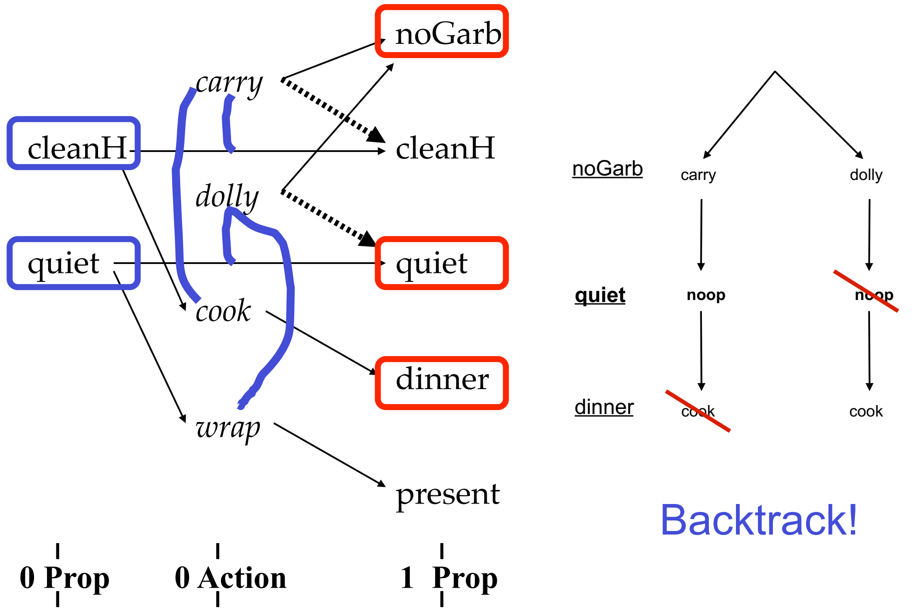
Search Action Layer 1 Again!



Search Action Layer 0

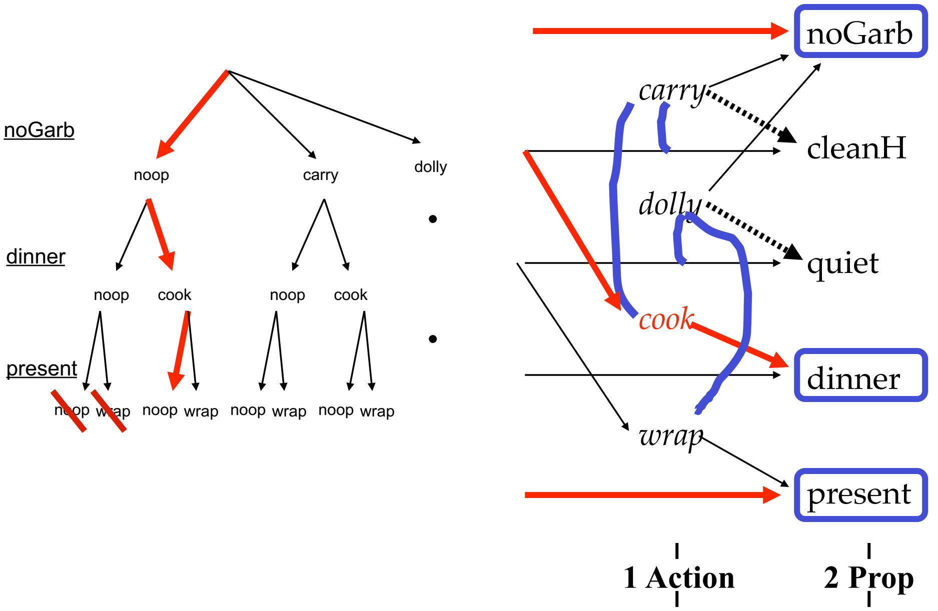


Search Action Layer 0

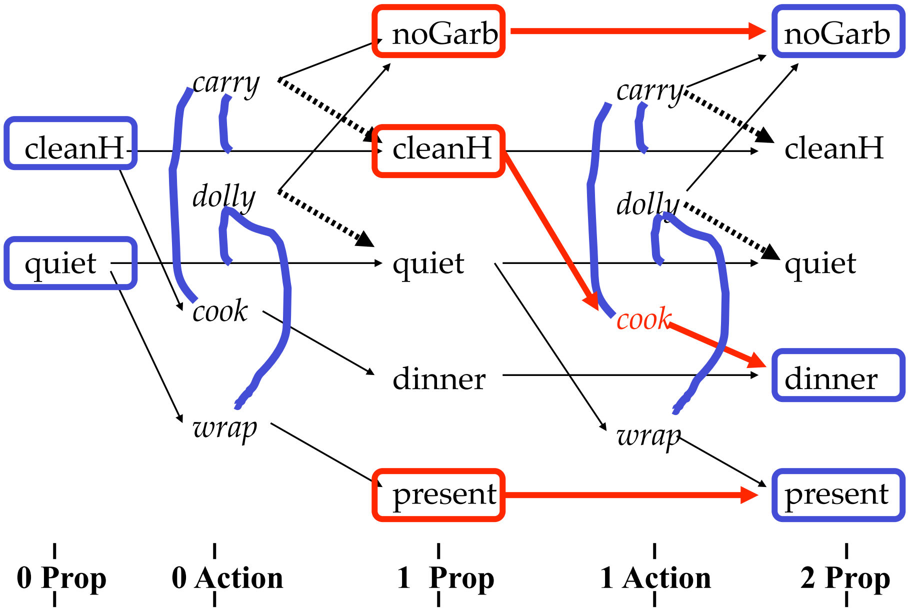


Backtrack!

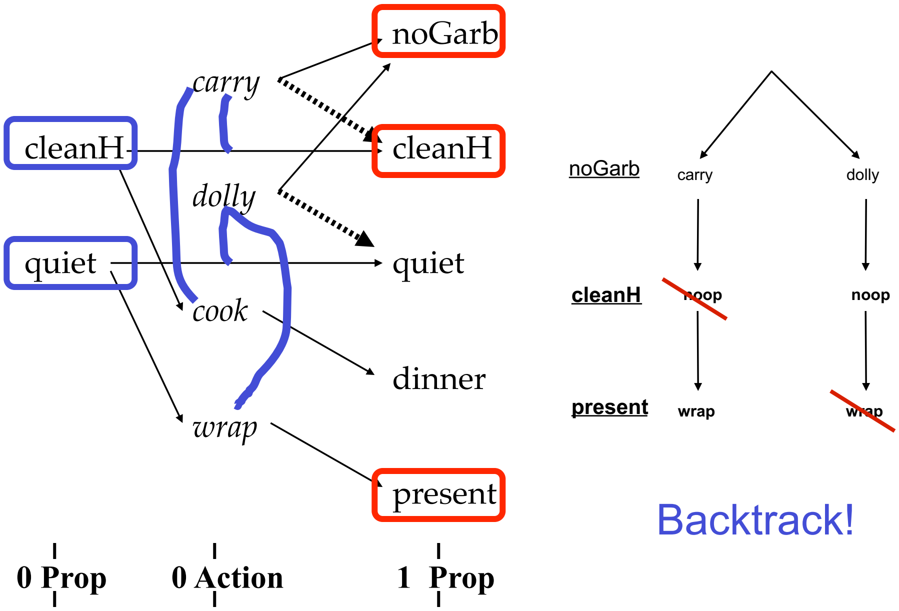
Search Action Layer 1 Again!



Search Action Layer 0

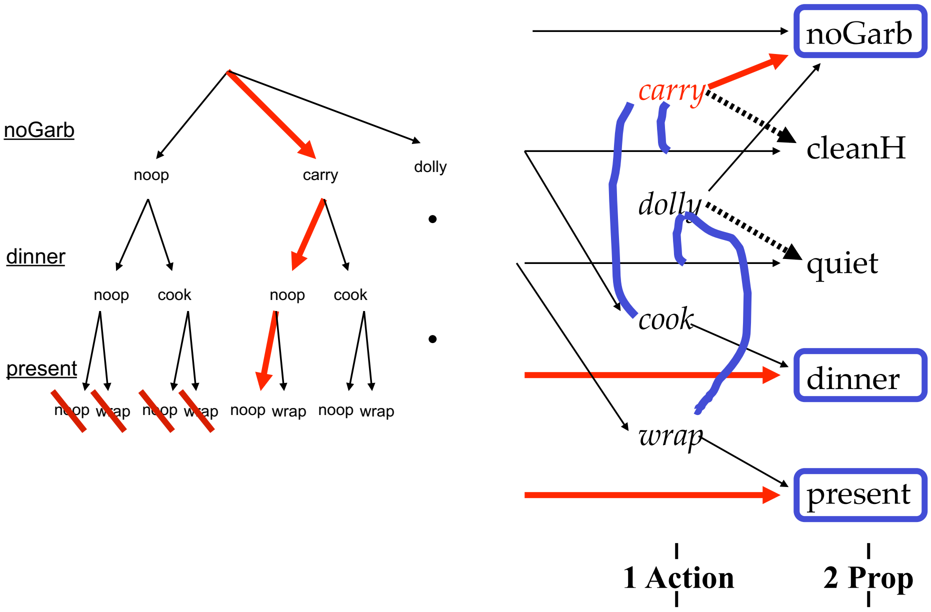


Search Action Layer 0

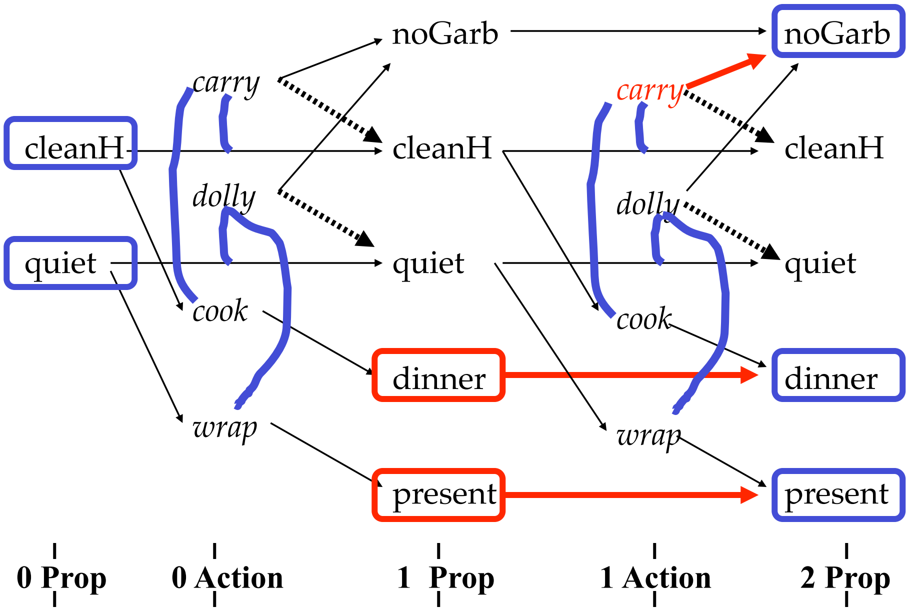


Backtrack!

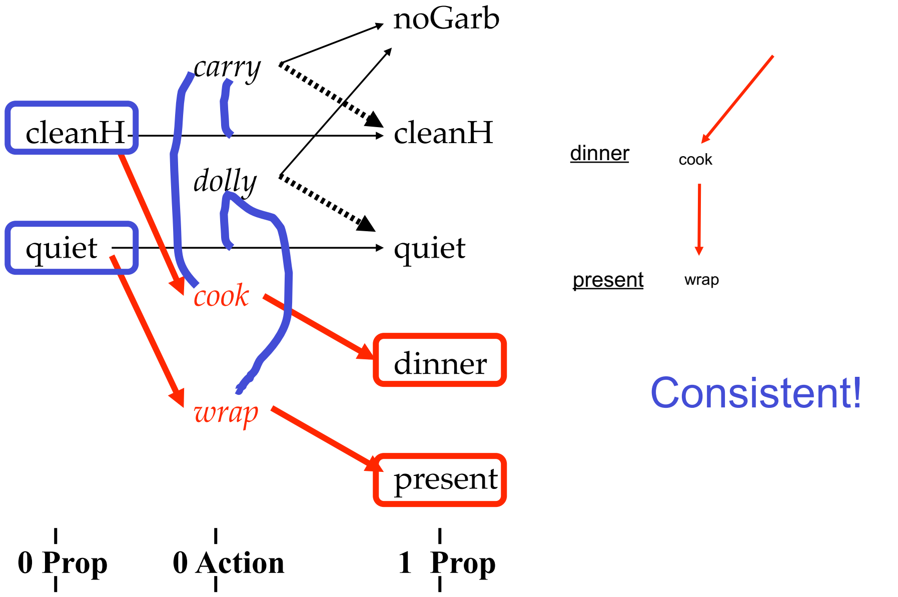
Search Action Layer 1 Again!



Search Action Layer 0

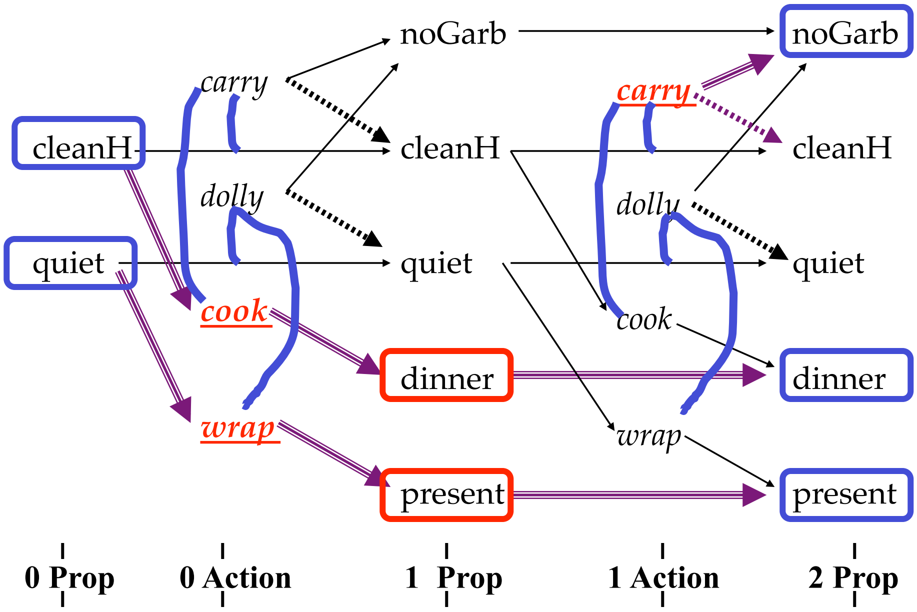


Search Action Layer 0



Consistent!

Solution: Cook & Wrap, then Carry



Outline

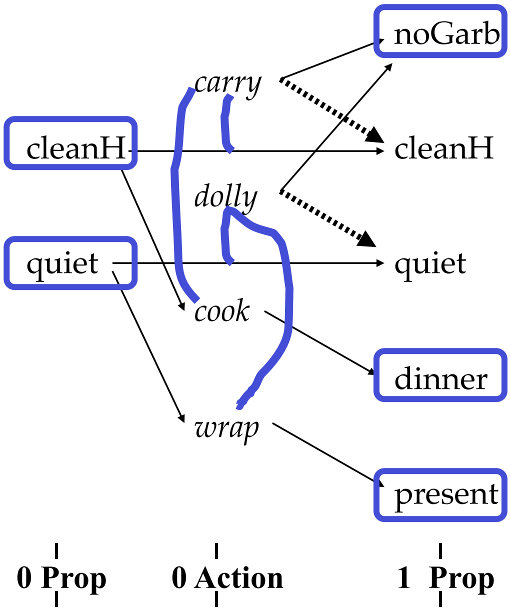
- Graph Plan
  - Solution Extraction
  - Memos
  - Properties
  - Termination with Failure

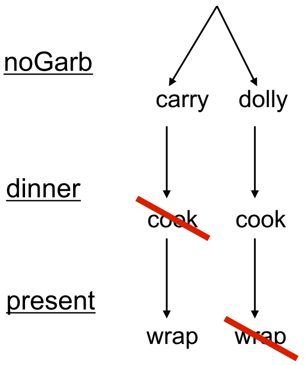
Memos of Inconsistent Subgoals

To prevent wasted search effort:

- If a goal set at layer k cannot be achieved,
  Then memoize the set at k (~ nogood / conflict).
- Check each new goal set at k against memos.
  - If memo,
    - Then fail,
    - Else test by solving a CSP.

Search Layer 0: Record Memo

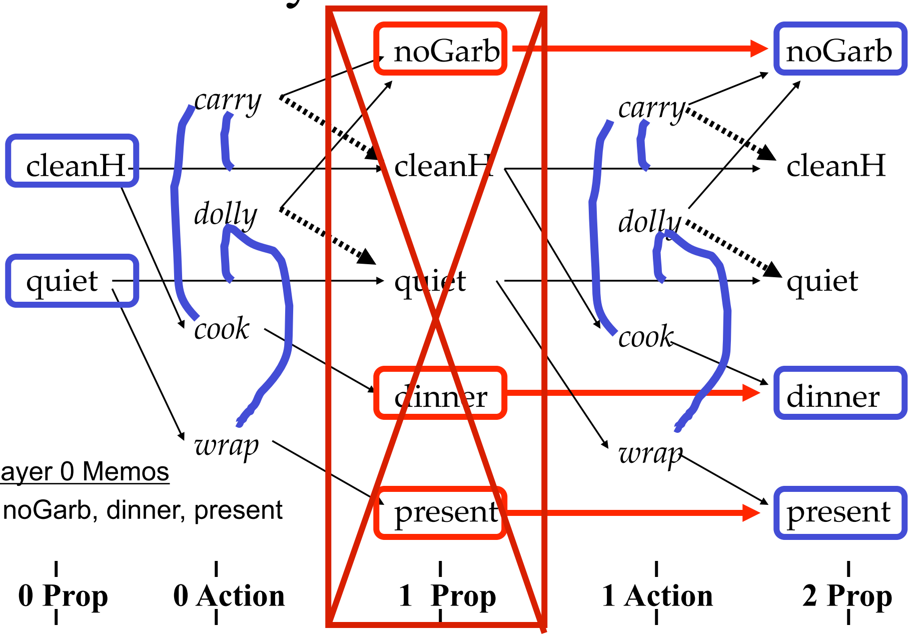




Layer 0 Memos

- noGarb, dinner, present

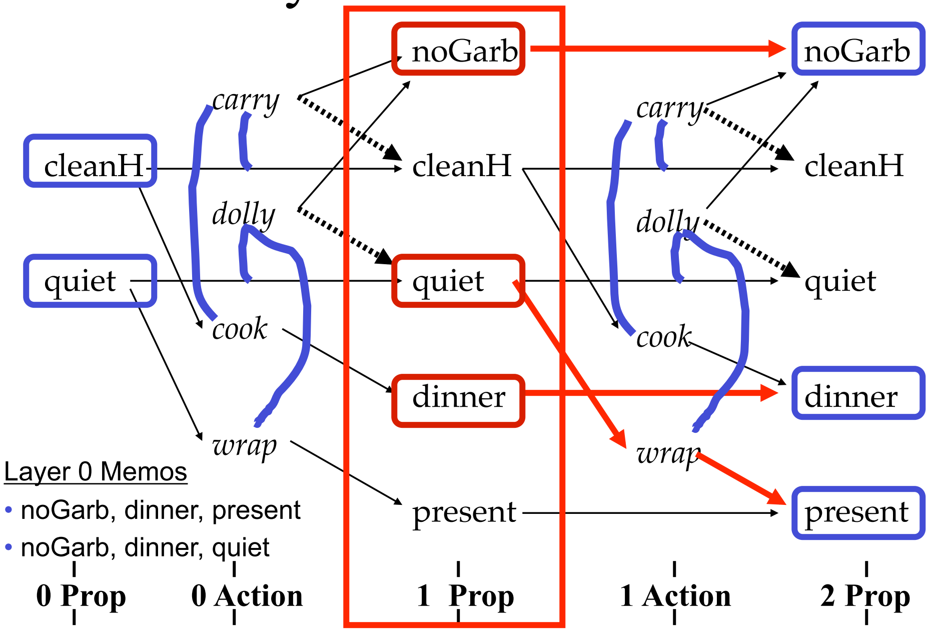
Search Layer 1: Check L0 memos



Layer 0 Memos

- noGarb, dinner, present

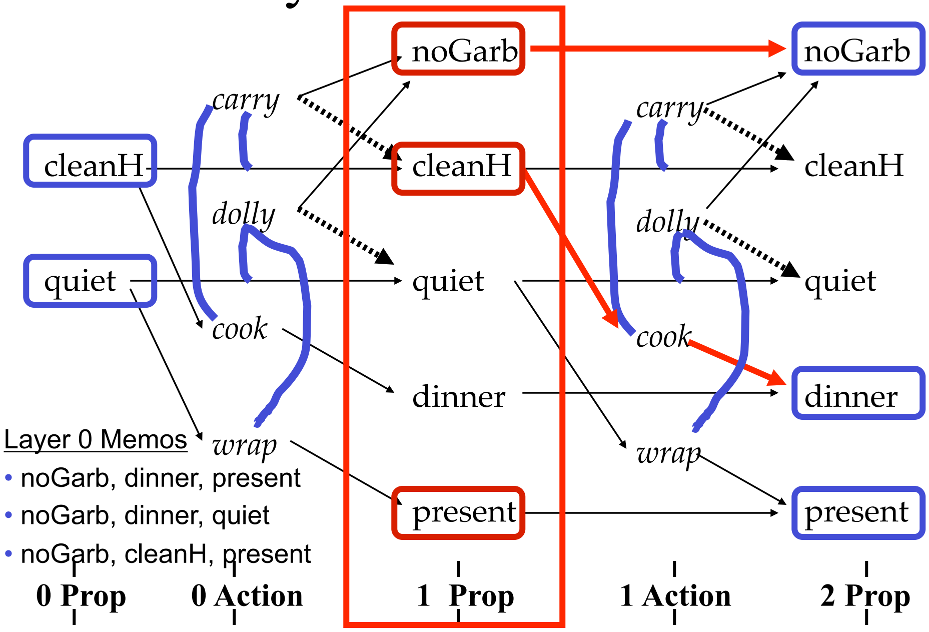
Search Layer 1: New Memo 2



Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet

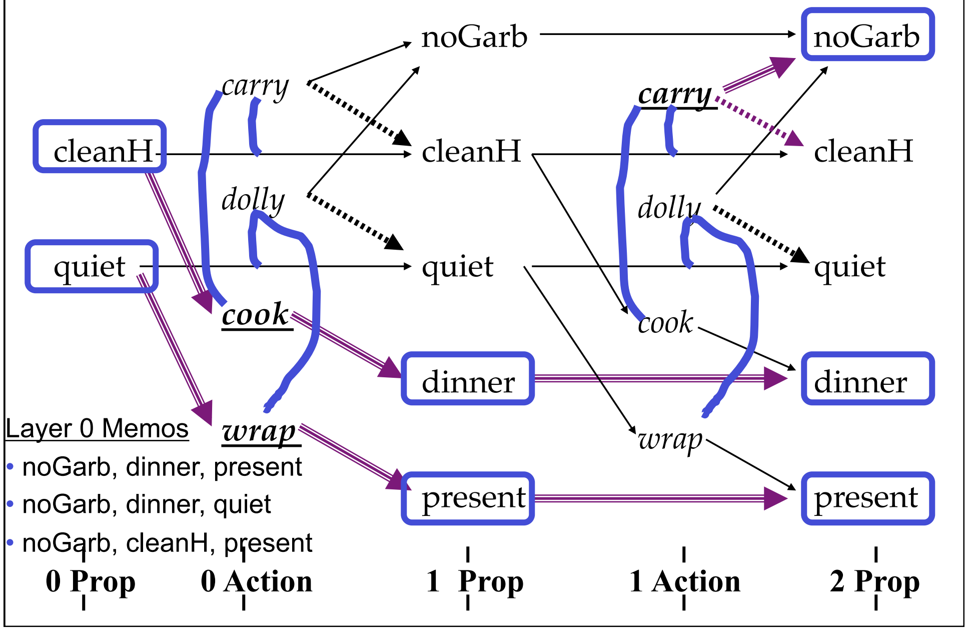
Search Layer 1: New Memo 3



Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet
- noGarb, cleanH, present

Solution Found: (Not a Memo)



noGarb noGarb

carry ***carry***

cleanH cleanH cleanH

dolly *dolly*

quiet quiet quiet

cook *cook*

dinner dinner

wrap *wrap*

present present

Layer 0 Memos

- noGarb, dinner, present
- noGarb, dinner, quiet
- noGarb, cleanH, present

0 Prop **0 Action** **1 Prop** **1 Action** **2 Prop**

Outline

- Review
- Graph Plan
  - Solution Extraction
  - Memos
  - Properties
  - Termination with Failure
- Execution
- Planning in a Continuous Domain for Deep Sea Exploration

Properties:
Optimality and Redundancy

- Plans guarantee parallel optimality.
  - Parallel plan will take as short a time as possible.
- Plans don' t guarantee sequential optimality.
  - Might be possible to achieve all goals
    at a later layer using fewer actions.
- Plans do not contain redundant steps.
  - Achieved by preferring no-ops.

Plan Graph Properties:
Fixed Points

- Propositions monotonically increase.
  - Once added to a layer they remain in successive layers.
- Mutexes monotonically decrease.
  - Once a mutex has decayed it never reappears.

➔The graph eventually reaches a fix point.

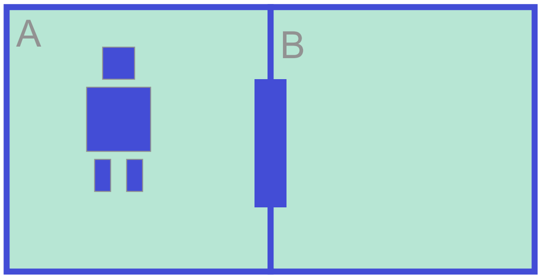
- Level where propositions and mutexes no longer change.

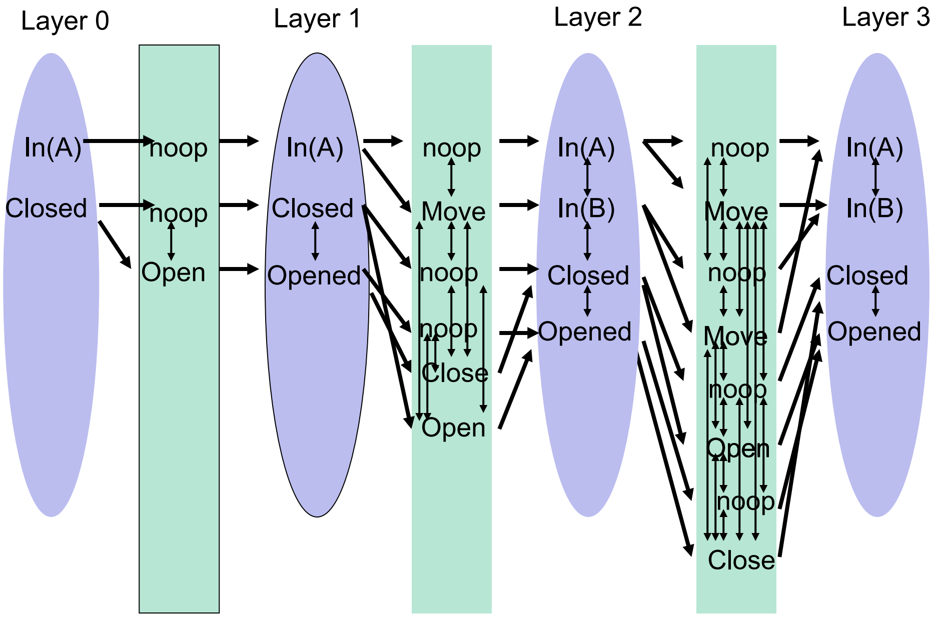
Fix point Example:
Door Domain

Move from room ?X to room ?Y
- pre: robot in ?X, door is open
- add: robot in ?Y
- del: robot in ?X

Open door
- pre: door closed
- add: door open
- del: door closed

Close door
- pre: door open
- add: door closed
- del: door open





Layer 3 is the fixed point of the graph – called “level out.”

Graph Search Properties

- Graphplan may need to expand well beyond the fix point to find a solution.

Why?

Gripper Example

Move from one room to another

- pre: robot in first room
- add: robot in second room
- del: robot in first room

Pick up ball

- pre: gripper free, ball in room
- add: holding ball
- del: gripper free, ball in room

Drop ball

- pre: holding ball, in room
- add: ball in room, gripper free
- del: holding ball

Gripper Example

- Fix point occurs at Layer 4.
  - All propositions concerning ball and robot locations are pairwise non-mutex after 4 steps.

- Solution layer depends on # balls moved.
  - E.g., for 30 balls,
    - solution is at layer 59;
    - 54 layers with identical propositions, actions and mutexes.

Outline

- Review
- Graph Plan
  - Solution Extraction
  - Memos
  - Properties
  - Termination with Failure

Termination Property

Graphplan returns failure if and only if no plan exists.

How?

Simple Termination

- If the fix point is reached and:
  - a goal is not asserted OR
  - two goals are mutex,

  Then return "No solution," without any search.
- Otherwise, there may be higher order exclusions (memos) that prevent a solution.

➔Requires a more sophisticated termination test.

Why Continue After FixPoint?

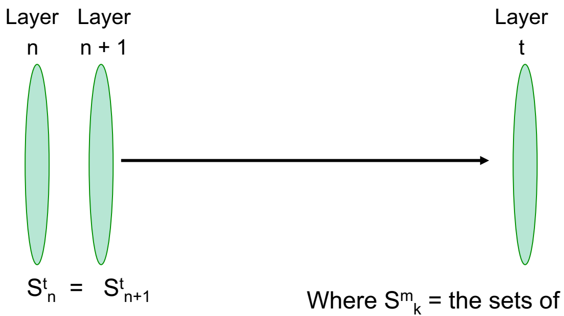
- Propositions, actions and mutexes no longer change after a fix point.
- But: memos (N-ary exclusions) do change.
  - New layers add time to the graph.
  - Time allows actions to be spaced so that memos decay.
  - Memos monotonically decrease.
    - Any goal set achievable at layer i, is achievable at i + n.

➔ Track memos & terminate on their fix point.

Appendix

Termination Test

- A graph “levels off” if the memos at layer n+1 are the same as at n.
- If the Graph levels off at layer n, and the current search stage is t >n, Then Graphplan can output "No Solution".



Layer n

Layer n + 1

Layer t

$S^t_n = S^t_{n+1}$

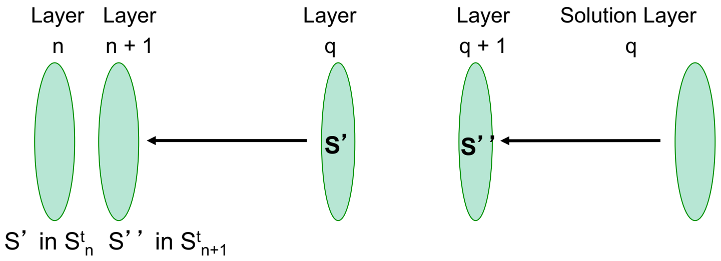
Where S^m_k = the sets of goals found unsolvable at layer k during search from m

Termination Property

- Theorem: Graphplan returns with failure iff the problem is unsolvable.
- Proof of “If the problem is unsolvable, then Graphplan returns with failure” : The number of goal sets found unsolvable at layer n from layer t will never be smaller than the number at n from layer t+1. In addition, there is a finite maximum number of goal sets. Hence, if the problem is unsolvable, eventually two successive layers will contain the same memoized sets.

If Graphplan outputs "No Solution," then the problem is unsolvable.

- Suppose the fix point is at layer n and Graphplan has completed an unsuccessful search starting at layer t > n.
- A plan to achieve any goal set that is unsolvable at layer n+1 must, one step earlier, achieve some set unsolvable at layer n.
- Suppose Graphplan returns "No Solution," but the problem is solvable:



Layer n

Layer n + 1

Layer q

Layer q + 1

Solution Layer q

S’

S’ ’

S’ in S^t_n S’ ’ in S^t_{n+1}

- If $S^t_n = S^t_{n+1}$ then S' and S" must both be in S^t_{n+1}. This means that some set in S^t_{n+1} will need to be achieved in n+1. this situation is contradictory.

16.410 / 16.413 Principles of Autonomy and Decision Making
Fall 2010