Activity Planning and Execution I: Operator-based Planning and Plan Graphs

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

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: Dechter, R., I. Meiri, J. Pearl, "Temporal Constraint Networks," Artificial Intelligence, 49, pp. 61-95,1991 posted on Stellar.
- Exam:
  - Mid-Term - October 20th.

Simple Spacecraft Problem

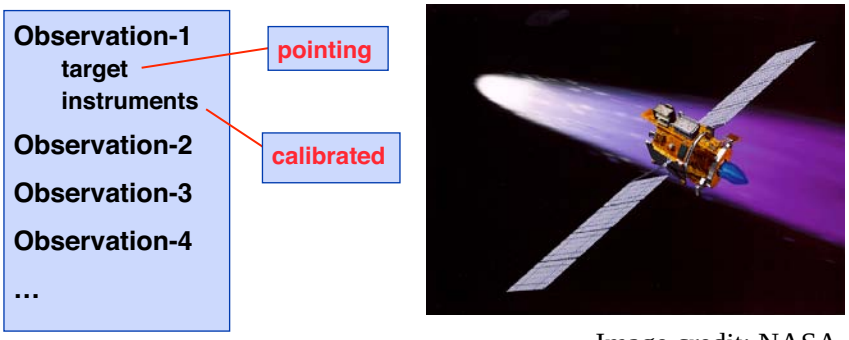


Image credit: NASA.

Propositions: Target Pointed To, Camera Calibrated?, Has Image?
Operators: Calibrate, Turn to Y, and Take Image.

Outline

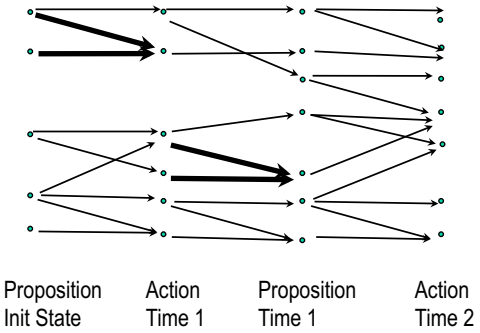
- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan Extraction

Graph Plan

- Developed in 1995 by Avrim Blum and Merrick Furst, at CMU.
- The Plan Graph compactly encodes all possible plans.
  - has been a key to scaling up to realistic problems.
- Plan Graph representation used for:
  - An encoding method for formulating planning as a CSP.
  - Relaxed planning as an admissible heuristic (state space search + A*).
- Approach has been extended to reason with temporally extended actions, metric and non-atomic preconditions and effects.

Approach: Graph Plan

1. Construct compact constraint encoding of state space from operators and the initial state.
   - *Planning Graph*
2. Generate plan by searching for a consistent subgraph that achieves the goals.



Representing States

- *State*
  - A consistent conjunction of propositions (positive literals).
    - E.g., (and (cleanhands) (quiet) (dinner) (present) (noGarbage))
    - All unspecified propositions are false.
- *Initial State*
  - Problem state at time i = 0.
    - E.g., (and (cleanHands) (quiet)).
- *Goal State*
  - A partial state.
    - E.g., (and (noGarbage) (dinner) (present)).
  - A Plan moves a system from its initial state to a final state that extends the goal state.

Representing Operators

(:operator cook :precondition (cleanHands)
:effect (dinner))

Note: STRIPS doesn't allow derived effects; you must be complete!

Preconditions: Propositions that must be true to apply the operator.
- A conjunction of propositions (no negated propositions).

Effects: Propositions that the operator changes, given that the preconditions are satisfied.
- A conjunction of propositions (called adds) and their negation (called deletes).

(Parameterized) Operator Schemata

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

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

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))

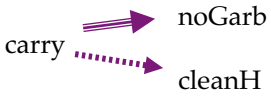
+ noops

Plan: (Cook, Wrap, Carry)

Visualizing Actions

(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))

(:operator **carry** *:precondition*
:effect (and (noGarbage) (not (cleanHands)))

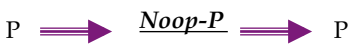


11

Visualizing Actions

- ***Persistence actions (No-ops)***
  - Every literal has a no-op action,
    which maintains it from time i to i+1.

    (:operator noop-P *:precondition* (P) *:effect* (P))



In Blum & Furst: (& lecture) Only persist positive literals.
AIMA: Persists negative literals as well.
either approach okay for PSet.

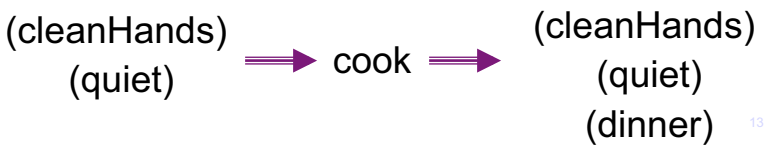
12

Operator Execution Semantics

If all propositions of :precondition appear in state i,
Then create state i+1 from i, by

- adding to i, all "add" propositions in :effects,
- removing from i, all "delete" propositions in :effects.

(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))



13

Operator Execution Semantics

If all propositions of :precondition appear in state i,
Then create state i+1 from i, by

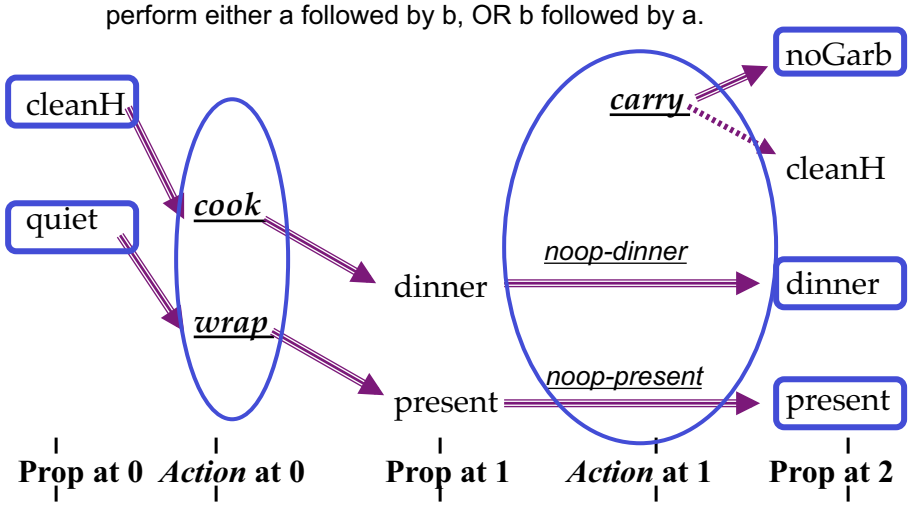
- adding to i, all "add" propositions in :effects,
- removing from i, all "delete" propositions in :effects.

(:operator **dolly** *:precondition*
:effect (and (noGarbage) (not (quiet)))

14

Representing Plans: <Actions[i] >

- Sets of concurrent actions that are performed at each time [i]
  - Concurrent actions can be interleaved in any order.
    - ⇨If actions a and b occur at time i, then it must be valid to perform either a followed by b, OR b followed by a.



A Complete Consistent Plan

Given an initial state that holds at time 0, and goal propositions, a plan is a solution iff it is:

Complete:

- The goal propositions all hold in the final state.

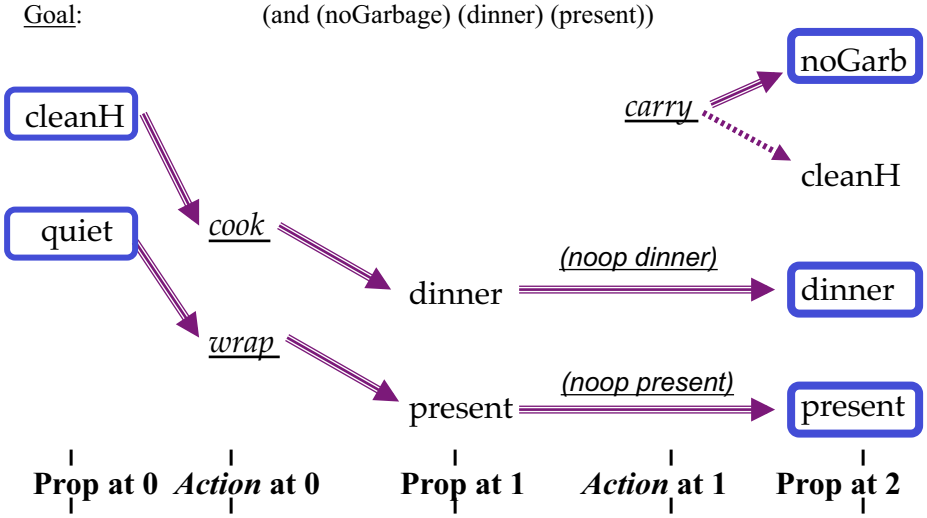
- The preconditions of every operator at time i, are satisfied by propositions at time i.

Consistent:

Example of a Complete Plan

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

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



A Complete Consistent Plan

Given an initial state that holds at time 0, and goal propositions, a plan is a solution iff it is:

Complete:

- The goal propositions all hold in the final state.
- The preconditions of every operator at time i, are satisfied by propositions at time i.

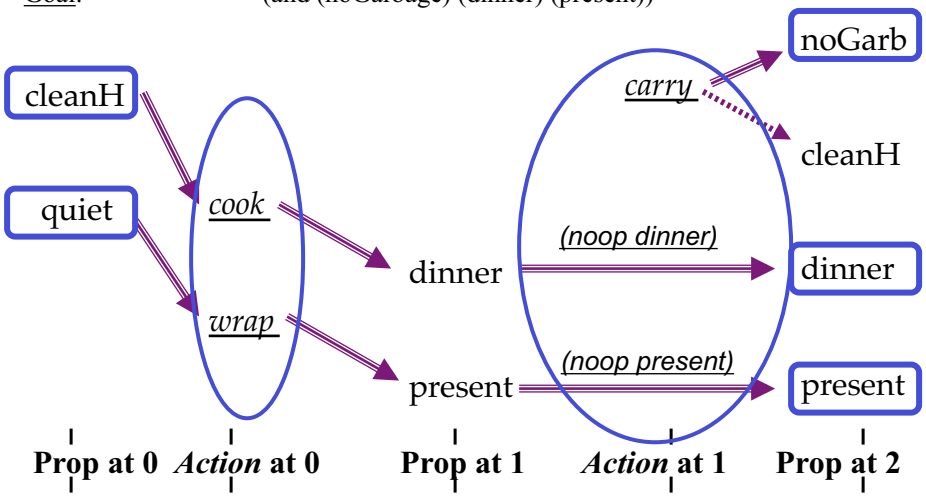
Consistent:

- The operators at any time i can be executed in any order, without one of these operators undoing:
  - the preconditions of another operator at time i.
  - the effects of another operator at time i.

Example of a Complete Consistent Plan

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

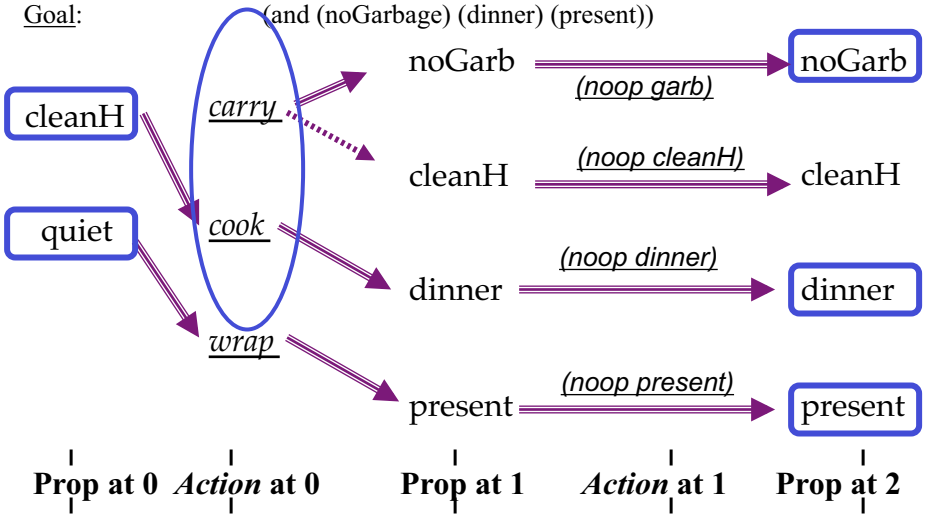
Goal: (and (noGarbage) (dinner) (present))



Example of a Complete Inconsistent Plan

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

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



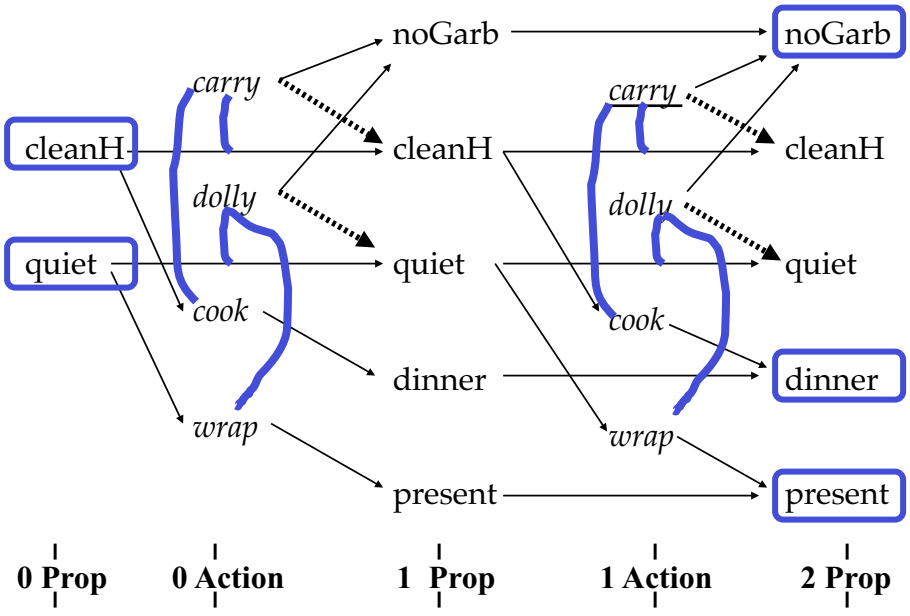
Outline

- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan Extraction

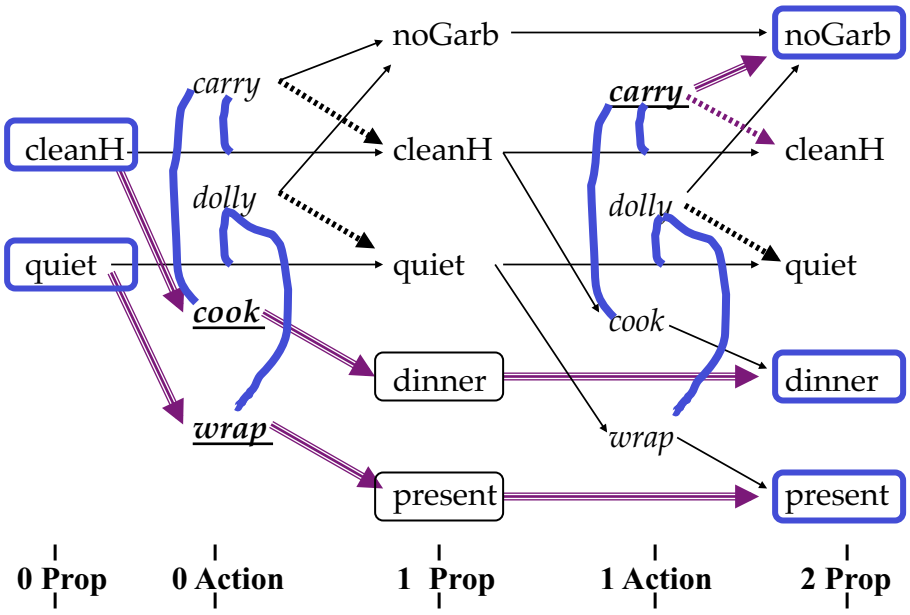
Graph Plan Algorithm

- Phase 1 – Plan Graph Expansion
  - Graph includes all plans that are complete and consistent.
  - Graph prunes many infeasible plans.
- Phase 2 - Solution Extraction
  - Graph frames a kind of constraint satisfaction problem (CSP).
  - Extraction selects actions to perform at each time point, by assigning variables and by testing consistency.

Example: Planning Graph and Solution



Example: Planning Graph and Solution



Graph Plan Algorithm

- Phase 1 – Plan Graph Expansion
  - Graph includes all plans that are complete and consistent.
  - Graph prunes many infeasible plans.
- Phase 2 - Solution Extraction
  - Graph frames a kind of constraint satisfaction problem (CSP).
  - Extraction selects actions to perform at each time point, by assigning variables and by testing consistency.
- Repeat Phases 1 and 2 for planning graphs with an increasing numbers of action layers.

Planning Graphs Prune

Initial state reachability:

Prunes partial states and actions at each time i that are not reachable from the initial state,

Consistency:

Prunes pairs of propositions and actions that are mutually inconsistent at time I, and

Goal state reachability:

plans that cannot reach the goals.

Graph Properties

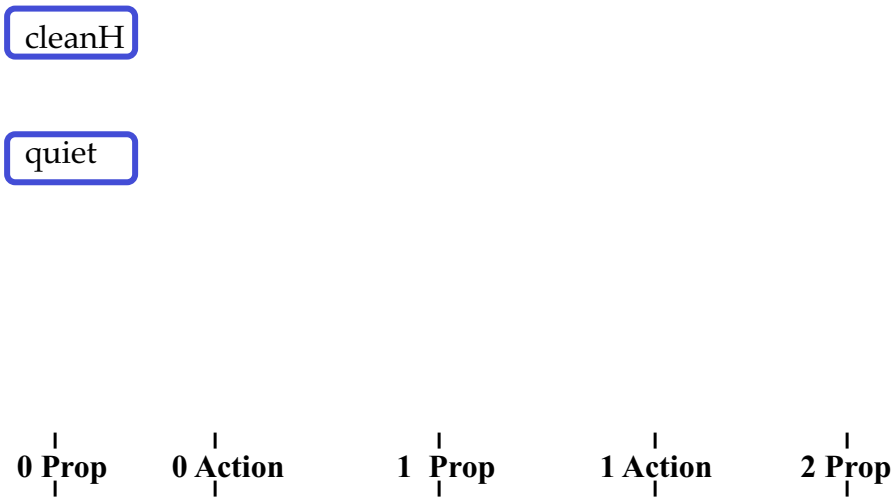
- Plan graphs are constructed in polynomial time and are of polynomial in size.
- Plan graphs do not eliminate all infeasible plans.

➔Plan generation requires *focused* search.

Constructing the Planning Graph… (Reachability)

- Initial proposition layer
  - Contains propositions that hold in the initial state.

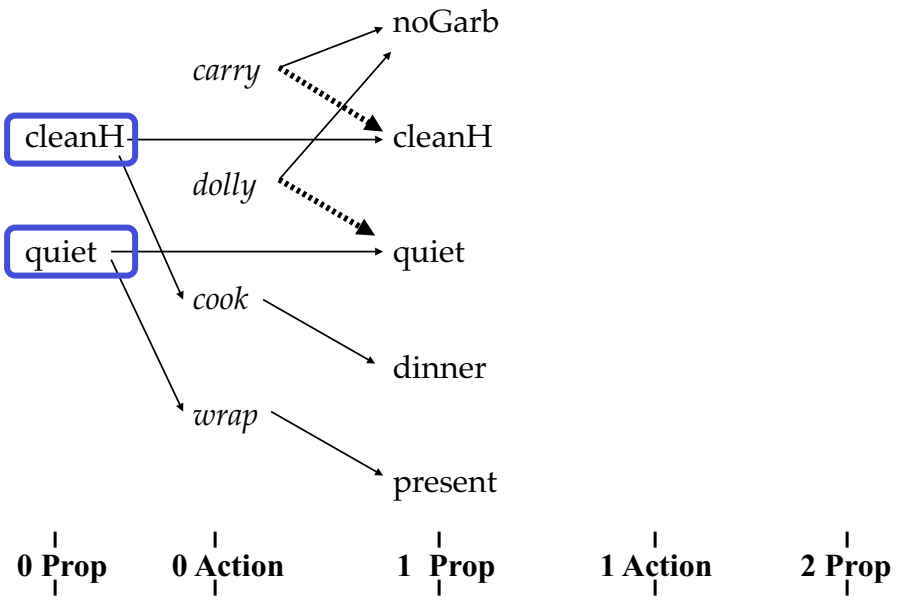
Example: Initial State, Layer 1



Constructing the Planning Graph… (Reachability)

- Initial proposition layer
  - Contains propositions that hold in the initial state.
- Action layer i
  - If all of an action's preconditions appear in proposition layer i,
  - Then add action to layer i.
- Proposition layer i+1
  - For each action at layer i,
  - Add all its effects at layer i+1.

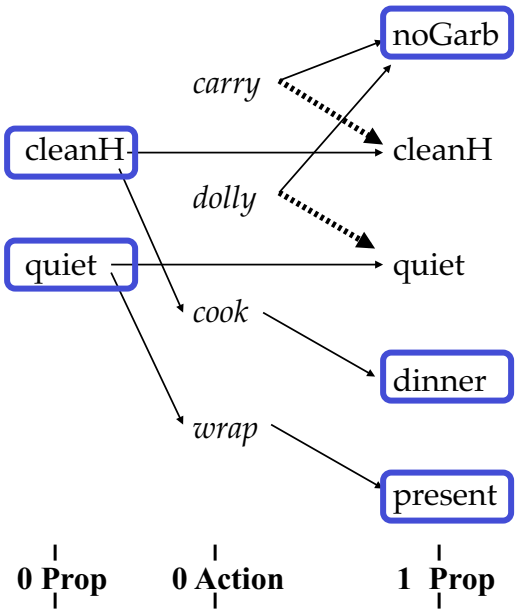
Example: Add Actions and Effects



Constructing the Planning Graph… (Reachability)

- Initial proposition layer
  - Contains propositions that hold in the initial state.
- Action layer i
  - If all of an action's preconditions appear in proposition layer i,
  - Then add action to layer i.
- Proposition layer i+1
  - For each action at layer i,
  - Add all its effects at layer i+1.
- Repeat adding layers until all goal propositions appear.

Round 1: Stop at Proposition Layer 1?



Do all goal propositions appear?

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

Constructing the Planning Graph… (Consistency)

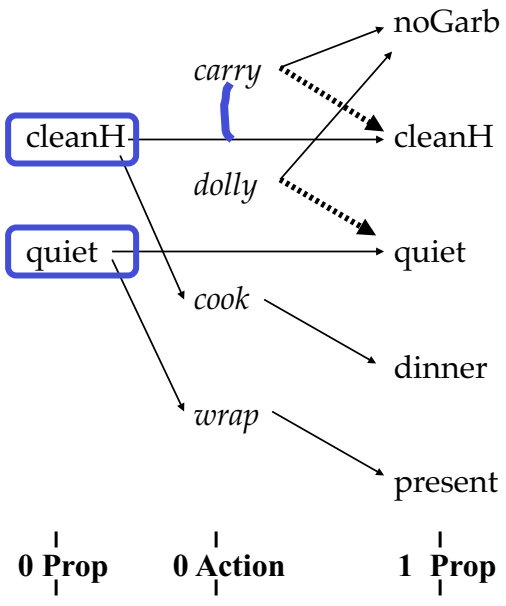
- Initial proposition layer
  - Contains propositions that hold in the initial state.
- Action layer i
  - If action's preconditions appear consistent in i [non-mutex],
  - Then add action to layer i.
- Proposition layer i+1
  - For each action at layer i,
  - Add all its effects at layer i+1.
- Identify mutual exclusions
  - Between actions in layer i, and
  - Between propositions in layer i + 1.
- Repeat until all goal propositions appear non-mutex.

Mutual Exclusion: Actions

- Actions A,B are **mutually *exclusive at level i*** if no valid plan could consistently contain both at i:
  - They have inconsistent effects.
    - A deletes B's effects.
  - Effects interfere with preconditions.
    - A deletes B's preconditions, or
    - vice-versa.
  - Their preconditions compete for needs.
    - A and B have inconsistent preconditions..

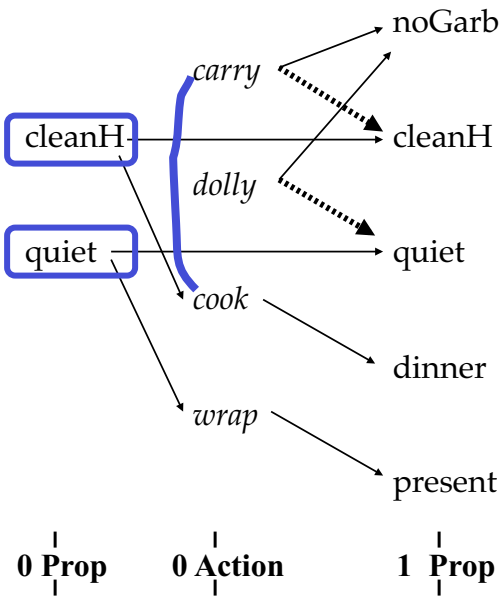
35

Mutual Exclusion: Actions



1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

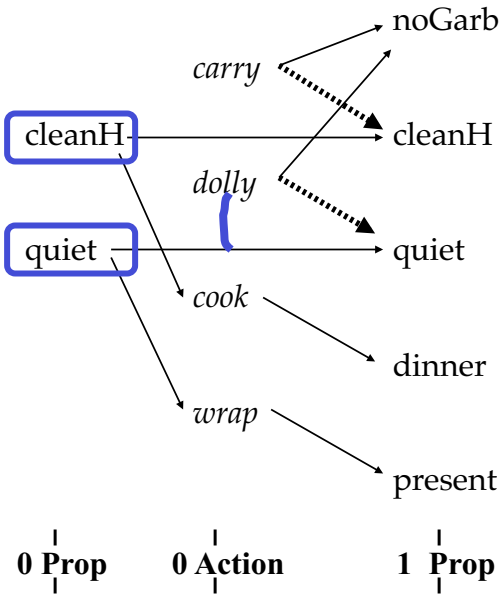
Mutual Exclusion: Actions



1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

0 Prop | 0 Action | 1 Prop | 1 Action | 2 Prop

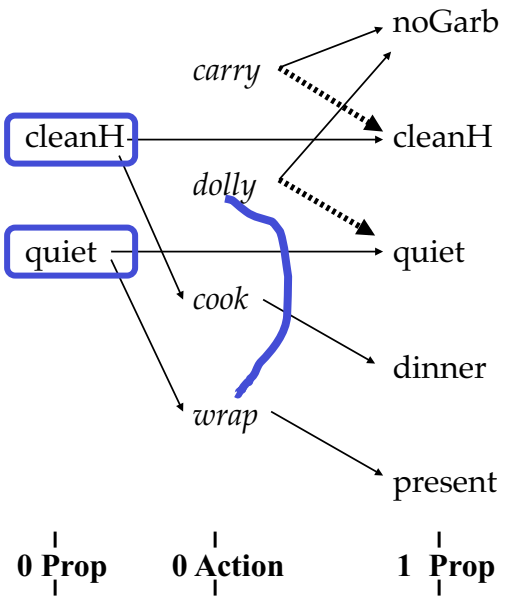
Mutual Exclusion: Actions



1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

0 Prop | 0 Action | 1 Prop | 1 Action | 2 Prop

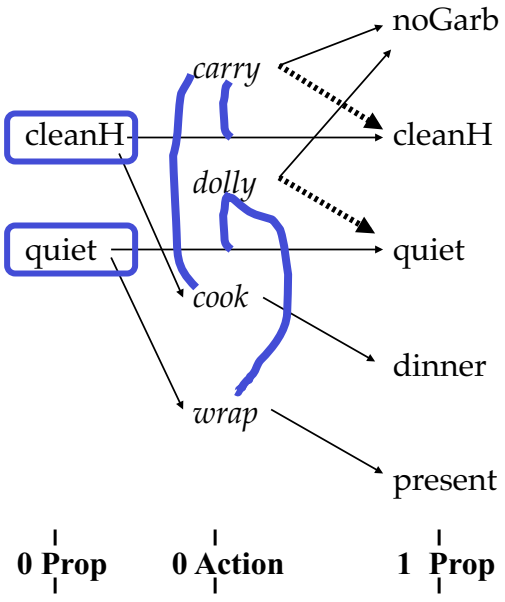
Mutual Exclusion: Actions



1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

0 Prop | 0 Action | 1 Prop | 1 Action | 2 Prop

Layer 1: Complete Action Mutexs



1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

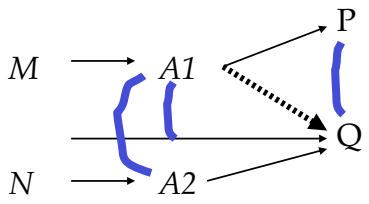
0 Prop | 0 Action | 1 Prop | 1 Action | 2 Prop

Mutual Exclusion: Proposition Layer

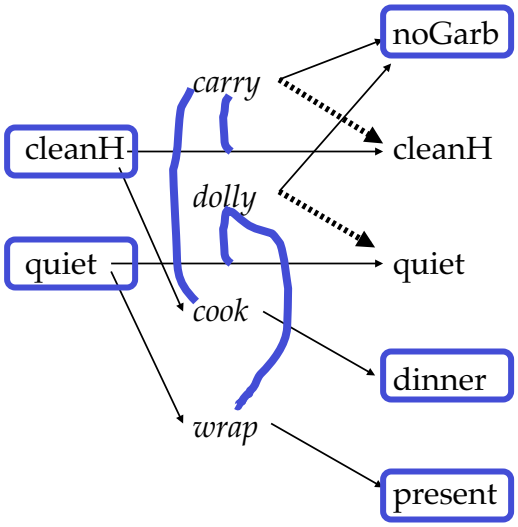
Propositions P,Q are ***inconsistent at i***

- if no valid plan could possibly contain both at i,

⇨ if at i, all ways to achieve P exclude each way to achieve Q.



Layer 1: Add Proposition Mutexs

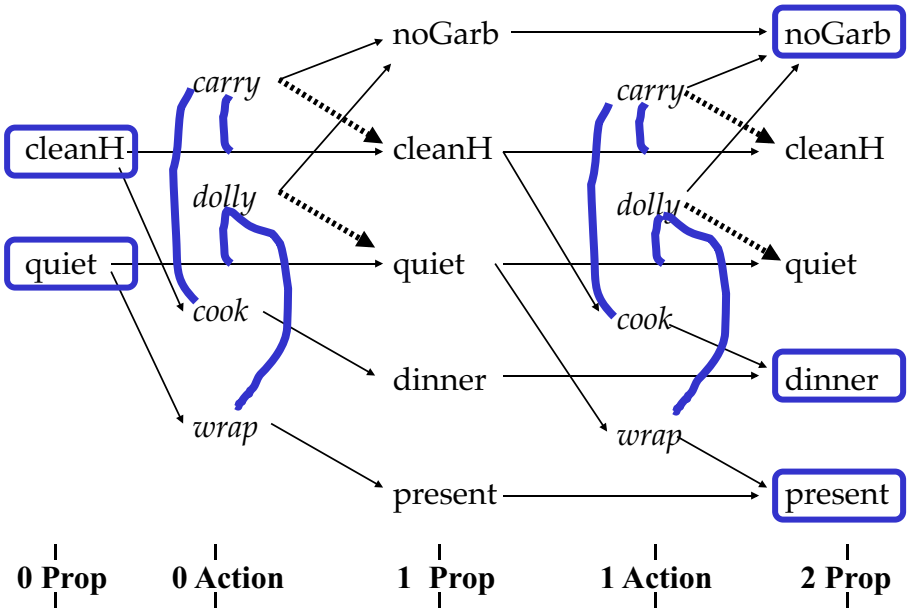


Do all goal propositions appear non-mutex?

No proposition mutexs.

0 Prop | 0 Action | 1 Prop | 1 Action | 2 Prop

Round 2: Extending The Planning Graph



Outline

- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan Extraction

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010