CS 4649/7649 Robot Intelligence: Planning

Constraints III: Activity Planning

CS 4649/7649 – Asst. Prof. Matthew Gombolay

Slides adapted from:
16.410 Brian Williams
Russell and Norvig AIMA

Assignments

- Due Wednesday, 9/09
  - Pset 3 due at 11:59 PM Eastern

- Due Monday, 9/14
  - Chapter 11 from Russel & Norvig

- Due Wednesday, 9/16
  - Pset 4 due at 11:59 PM Eastern

Outline

- Graph Plan
  - –Problem Statement
  - –Planning Graph Construction
  - –Plan Extraction
  - –Memos
  - –Properties
  - –Termination with Failure

Graph Plan

- Developed in 1995 by Avrim Blum and Merrick Furst, at CMU.

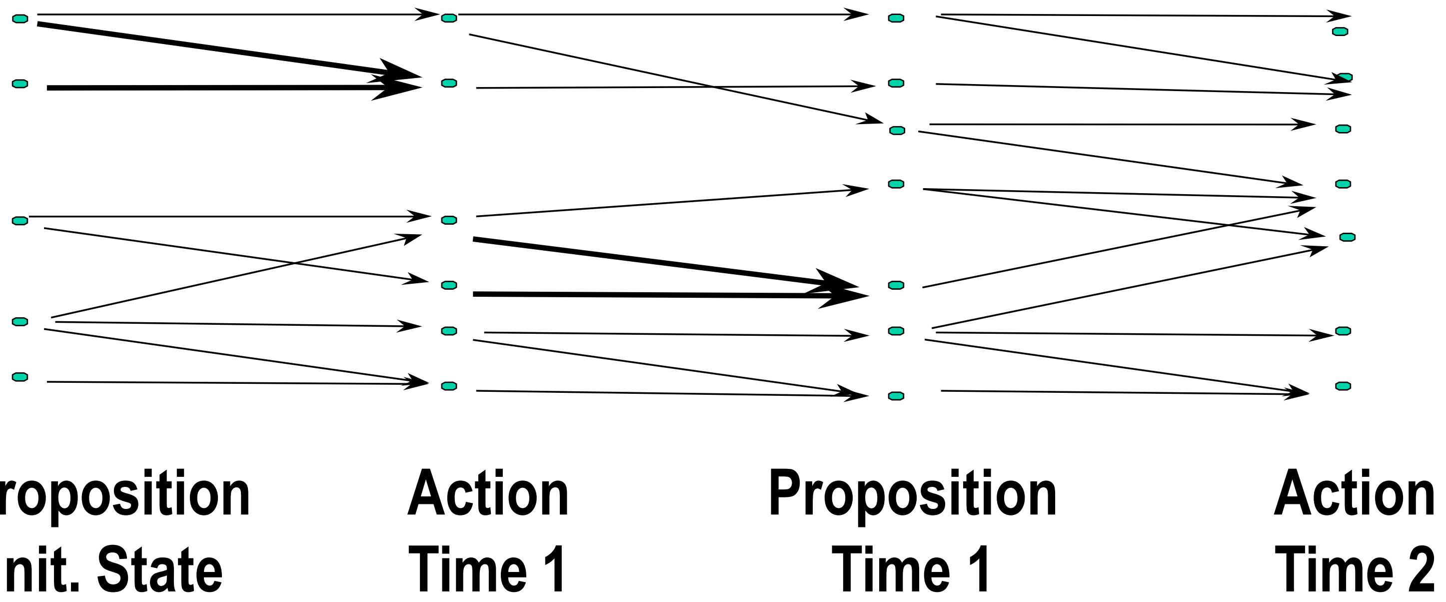
- The Plan Graph compactly encodes all possible plans.
  - –has been a key to scaling up to realistic problems.

- 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

Note: STRIPS (among other techniques) does not allow for “derived effects”; you must enumerate effects completely!

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

E.g., the computer doesn’t understand that cooking makes dinner

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

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))

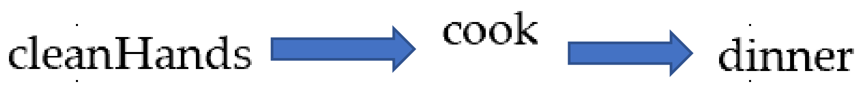
(:operator **wrap** *:precondition* (quiet)
:effect (present))

\+ noops

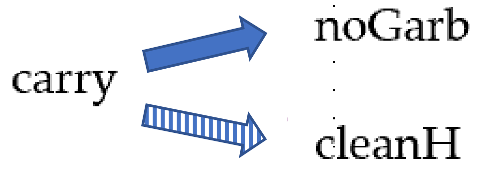
Plan: (Cook, Wrap, Carry)

Visualizing Actions

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



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



Dashed arrows indicates "delete"

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))

P → ***Noop-P*** → P

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)))

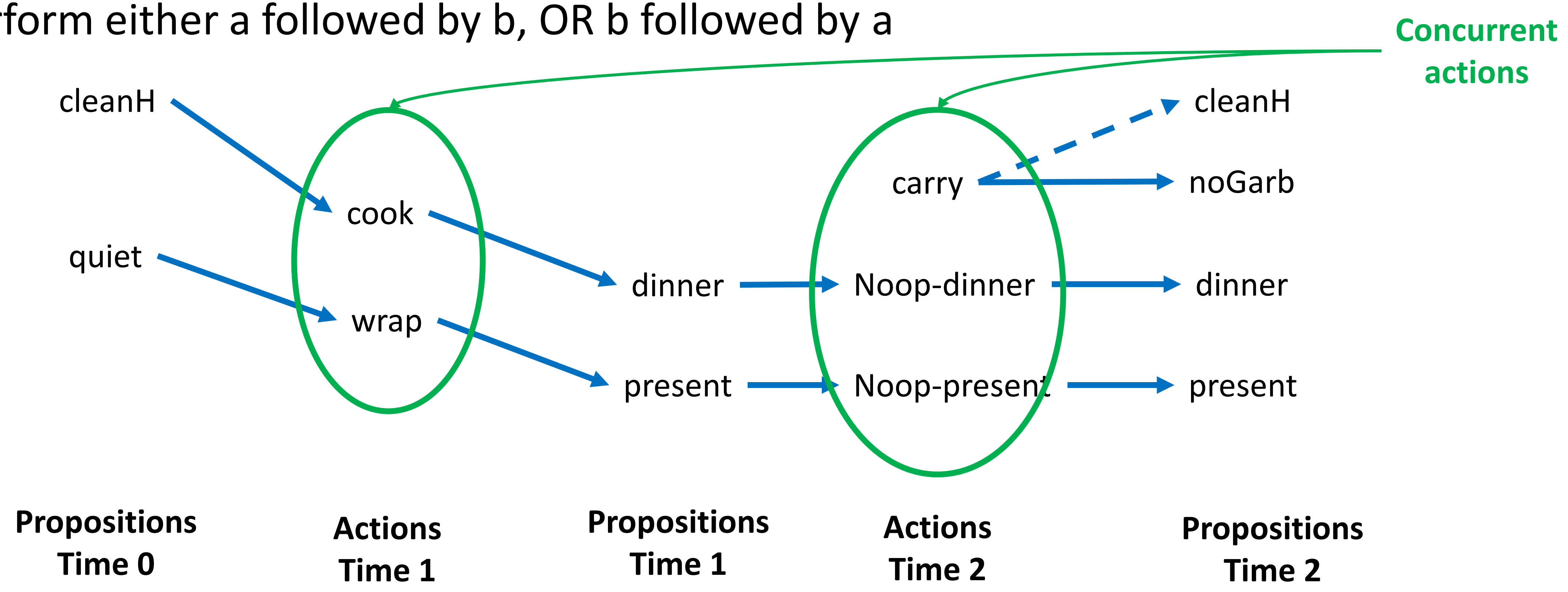
(cleanHands)
(quiet) → dolly → (cleanHands)
(noGarbage)

Representing Plans:

<u>Sets of concurrent actions that are performed at each time [i]</u>

- Concurrent actions can be interleaved in any order.

→ 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:

- Consistent:

A Complete, Consistent Plan

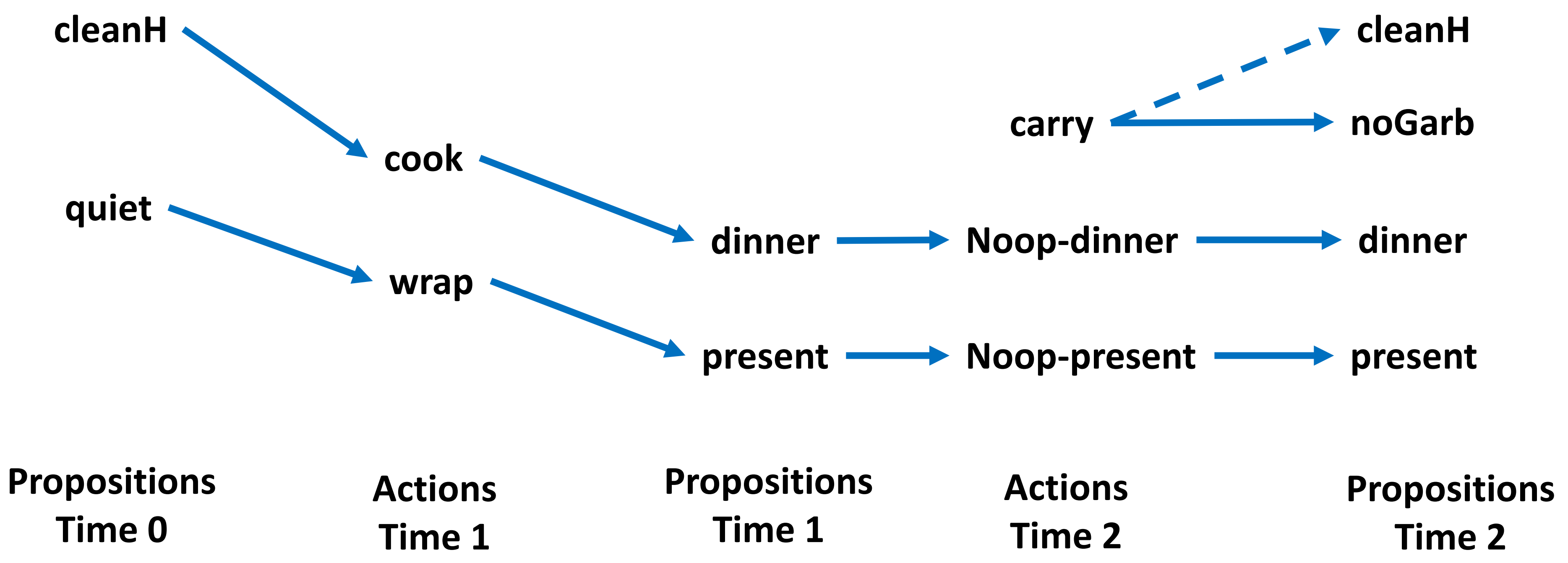
Given an initial state that holds at time 0, and goal propositions, a plan is a solution *iff* it is:

- 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

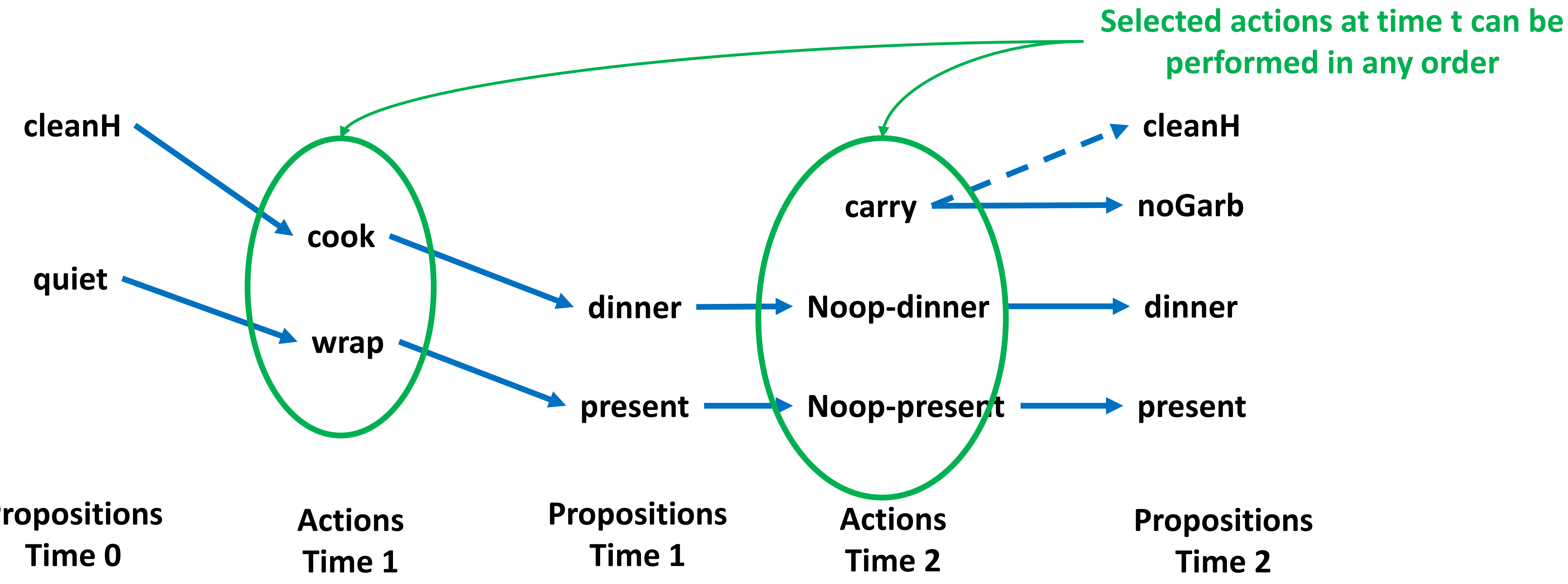
Given an initial state that holds at time 0, and goal propositions, a plan is a solution *iff* it is:

- 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))

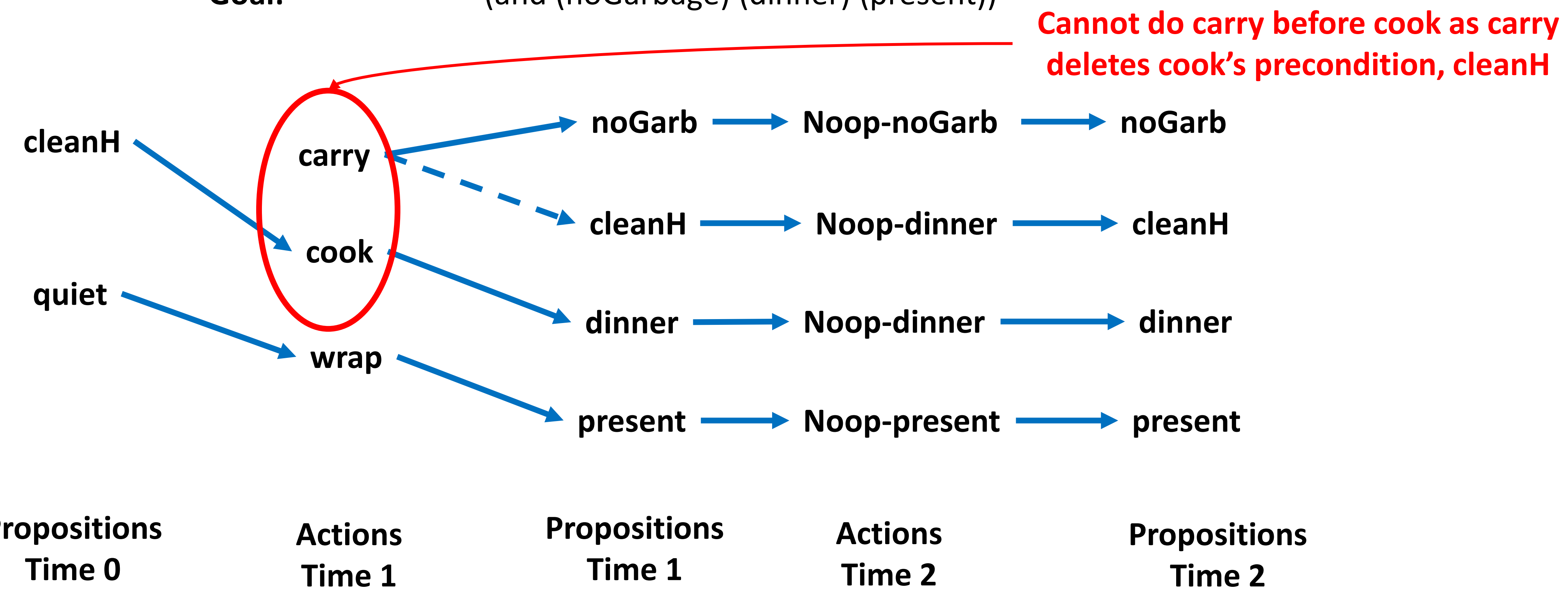
Goal: (and (noGarbage) (dinner) (present))



Example of a Complete, Inconsistent Plan

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

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



Outline

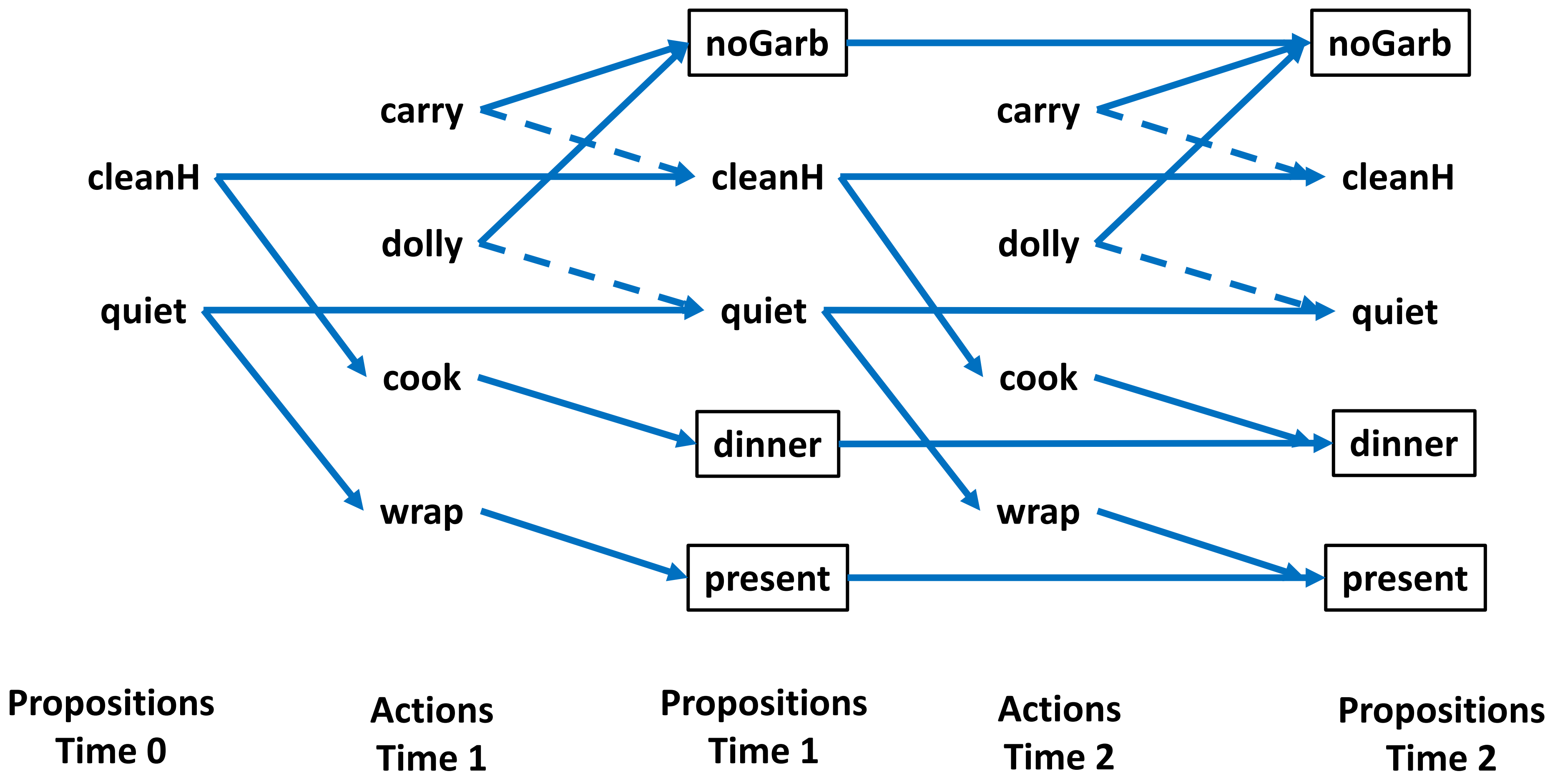
- Graph Plan
  - –Problem Statement
  - –Planning Graph Construction
  - –Plan Extraction
  - –Memos
  - –Properties
  - –Termination with Failure

Graph Plan Algorithm

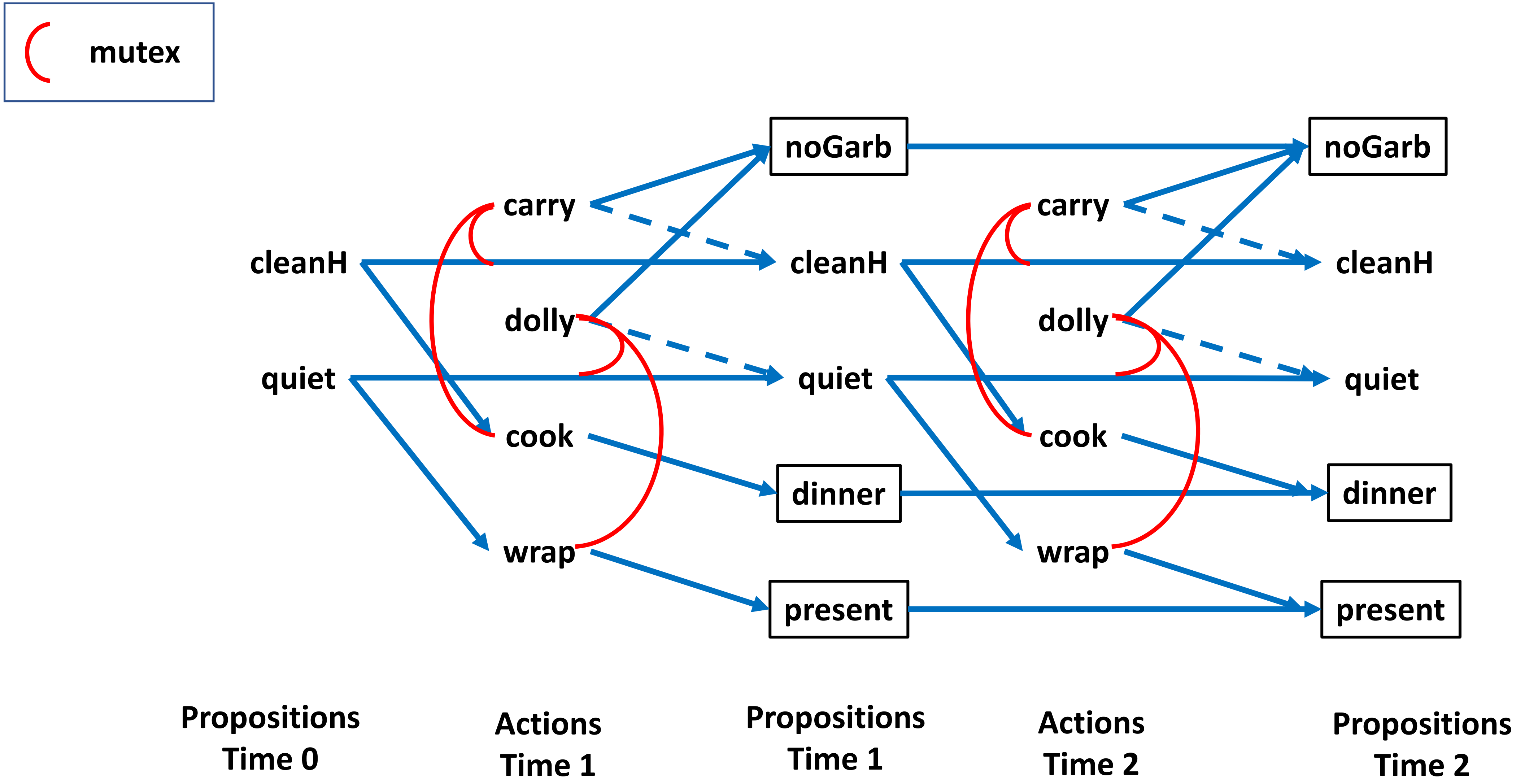
- Phase 1 – Plan Graph Expansion
  - Graph includes all plans that are complete and consistent.
  - Graph prunes many infeasible plans.

- 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.

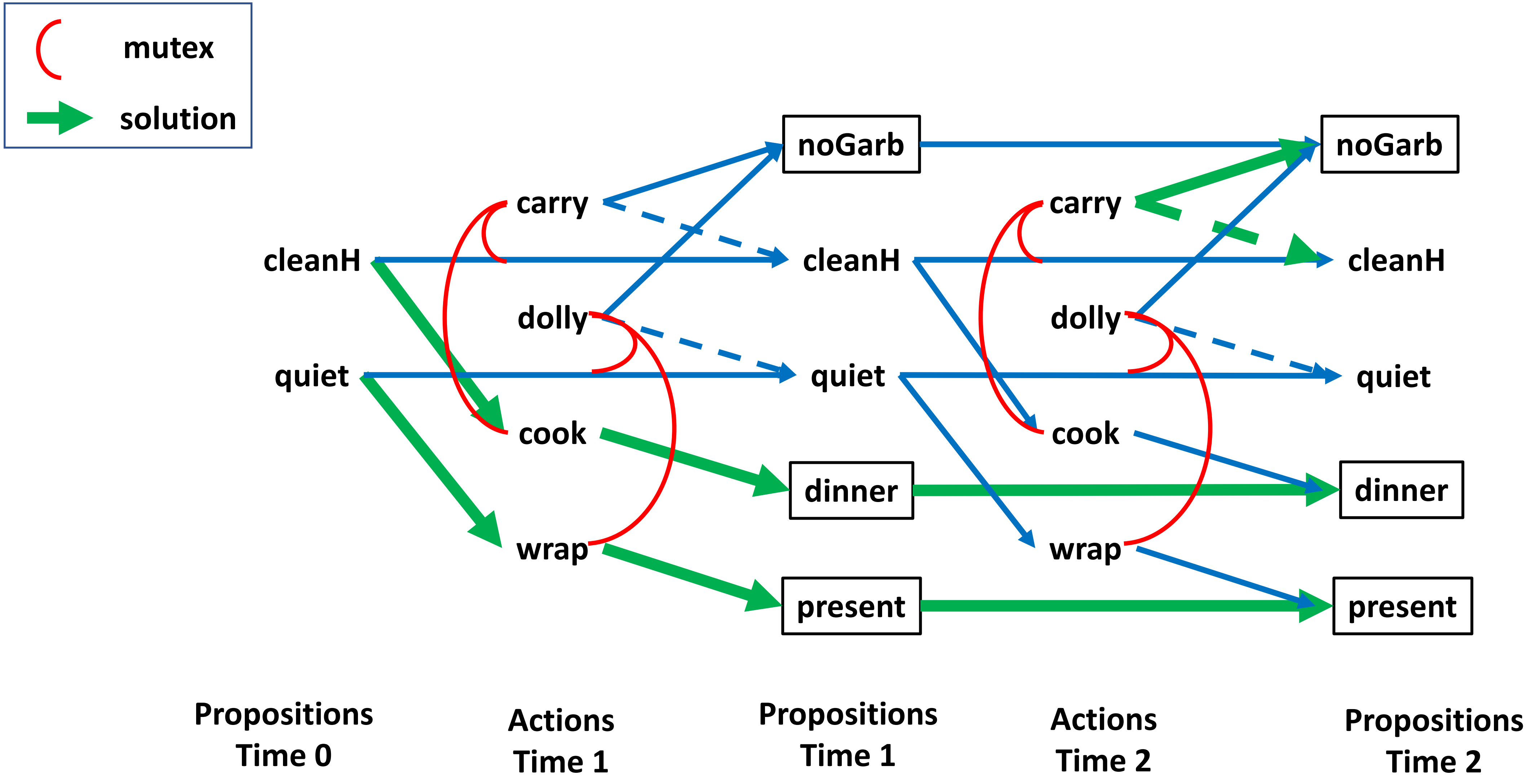
Example: Planning Graph and Solution



Example: Planning Graph and Solution



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

(mutex

cleanH

quiet

Propositions Time 0

Actions Time 1

Propositions Time 1

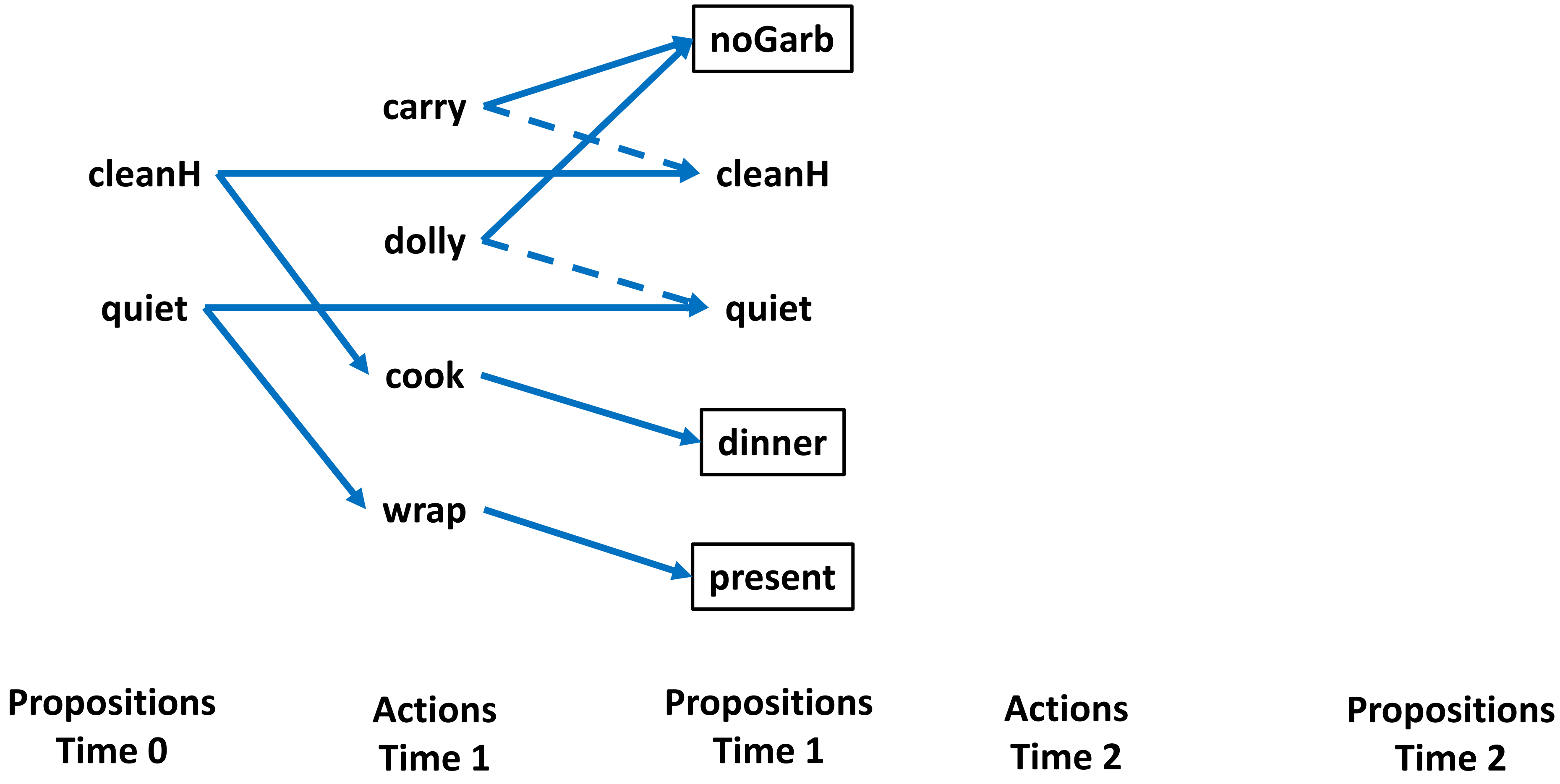
Actions Time 2

Propositions Time 2

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

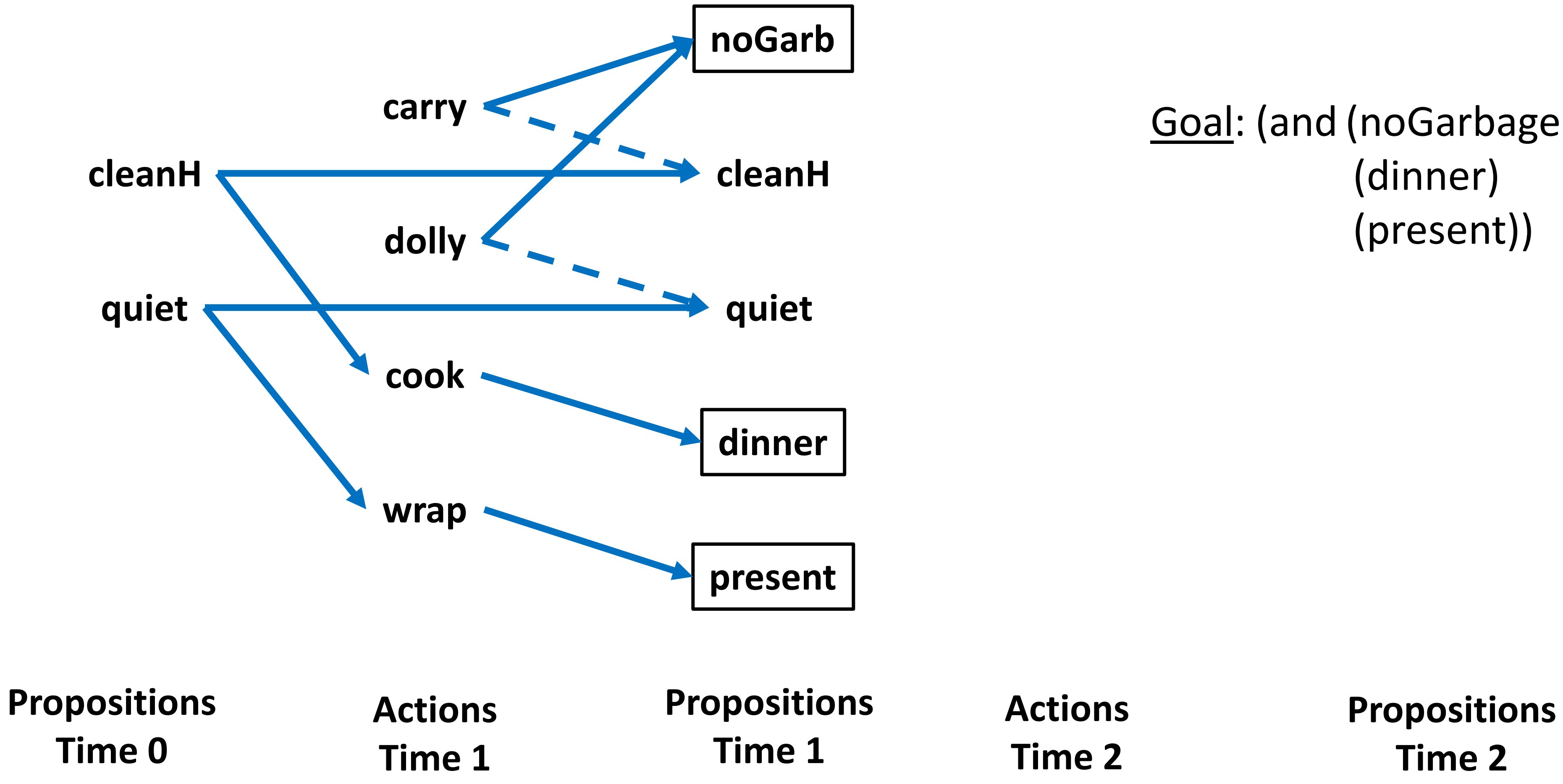


Constructing the Planning Graph… (Reachability)

- 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?



<u>Goal</u>: (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.

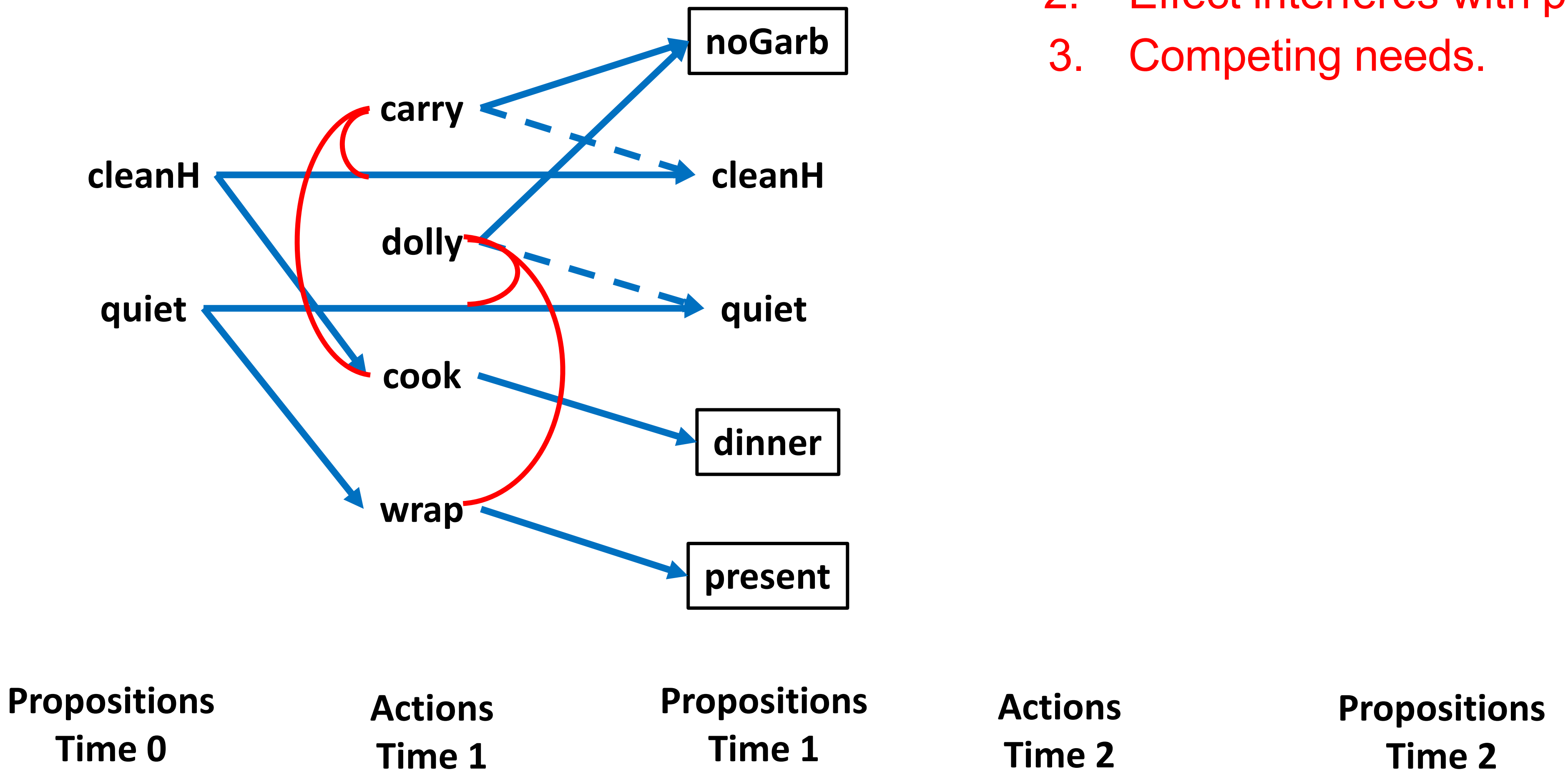
Mutual Exclusion: Actions

- 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..

Layer 1: Complete Action Mutexs



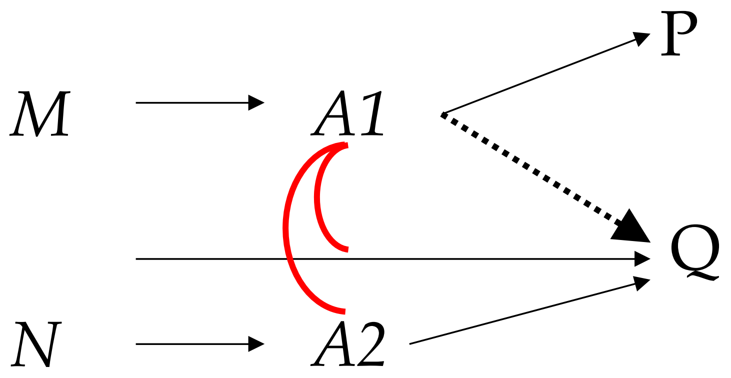
1. Inconsistent effects.
2. Effect interferes with precondition.
3. Competing needs.

Mutual Exclusion: Proposition Layer

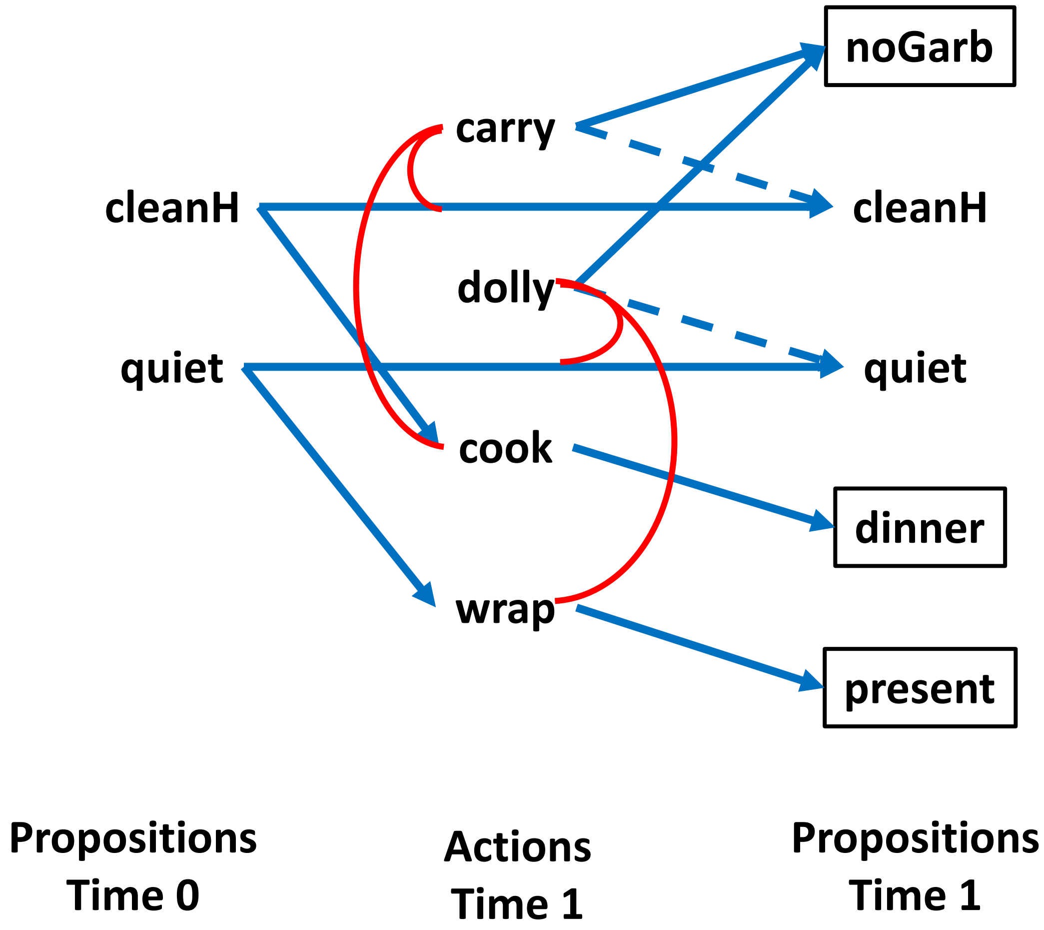
Propositions P,Q are ***inconsistent at i***

if no valid plan could possibly contain both at i,

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



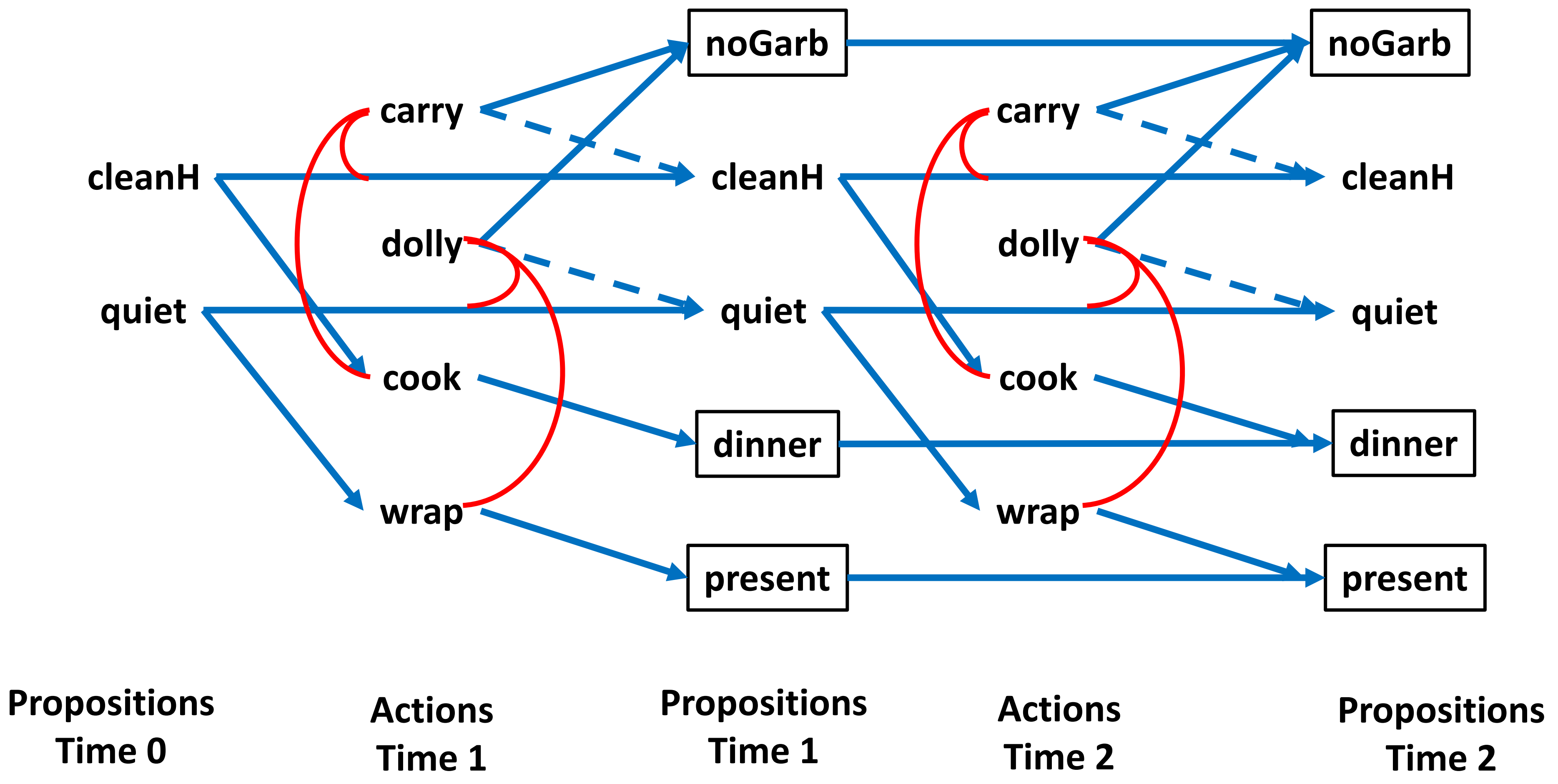
Layer 1: Add Proposition Mutexs



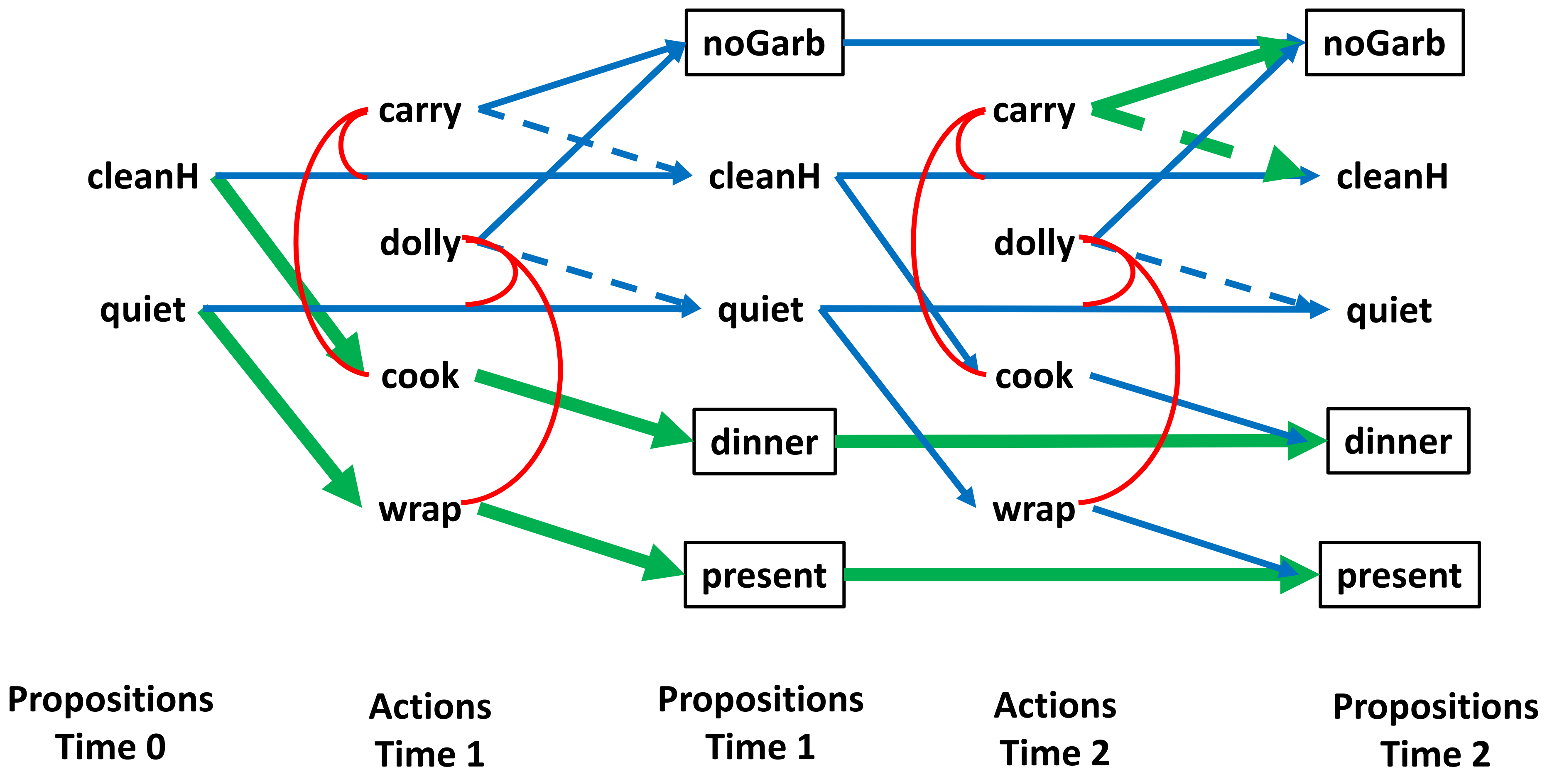
Do all goal propositions appear non-mutex?

No proposition mutexs.

Round 2: Extending The Planning Graph



Search Graph for Solution



Mid-lecture break

BBC | Sign in | Home | News | Sport | Reel

NEWS

Home | US Election | Coronavirus | Video | World | US & Canada | UK | Business | Tech | Science | Stories

Tech

Tesla Autopilot crash driver 'was playing video game'

26 February

REUTERS

The driver of the Tesla Model X died shortly after the crash

An Apple employee who died after his Tesla car hit a concrete barrier was playing a video game at the time of the crash, investigators believe.

The US National Transportation Safety Board (NTSB) said the car had been driving semi-autonomously using Tesla's Autopilot software.

Outline

- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan Extraction
  - Memos
  - Properties
  - Termination with Failure

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.

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.

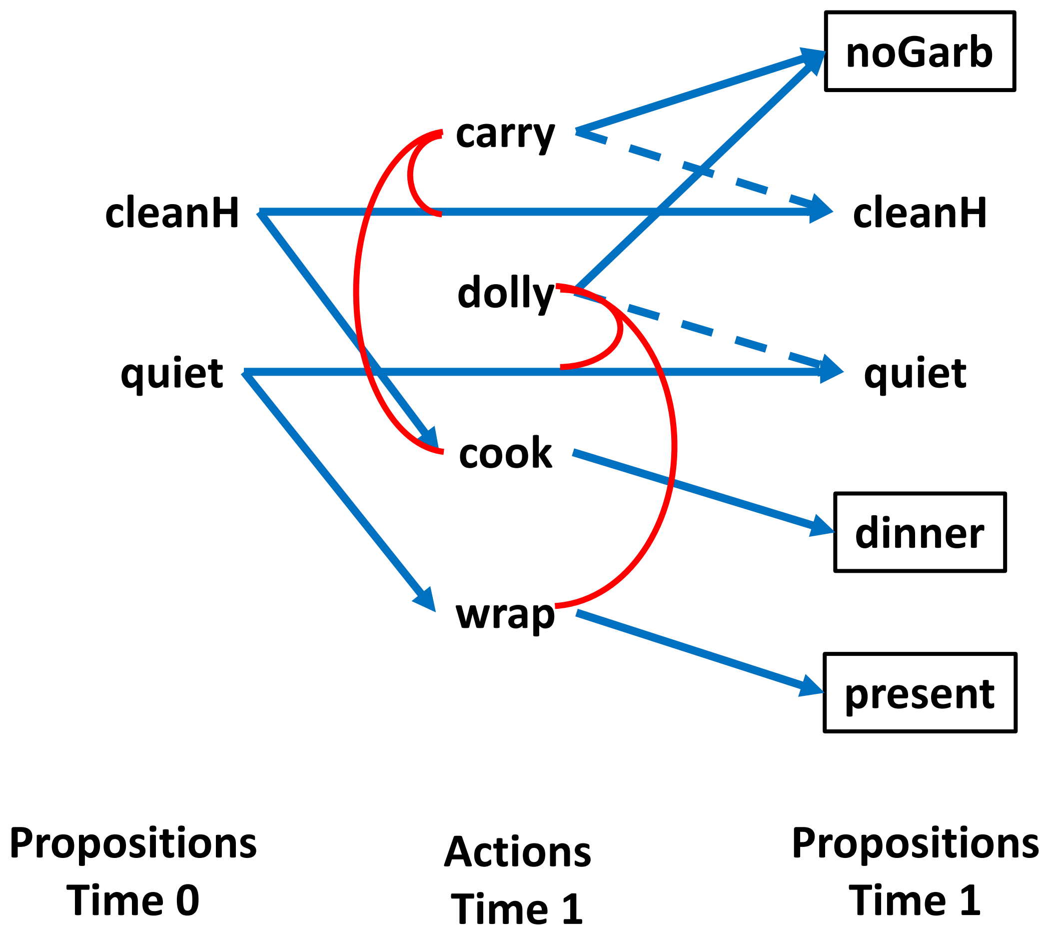
- Finally
  - If actions are found for all goals at time t,
  - Then recurse on t-1, using the action preconditions as goals,
  - Else backtrack to the next candidate solution at t+1.
  - Return plan if t = 0.

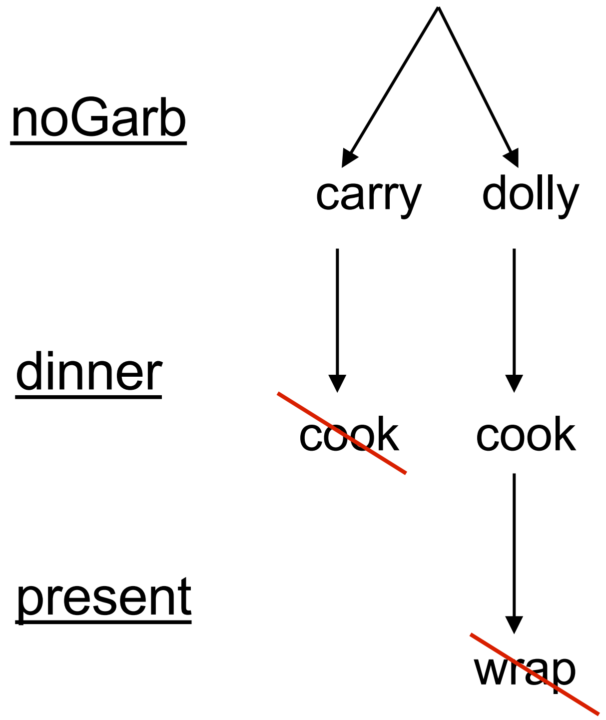
2. Search for a Solution (Alternatively)

- 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.

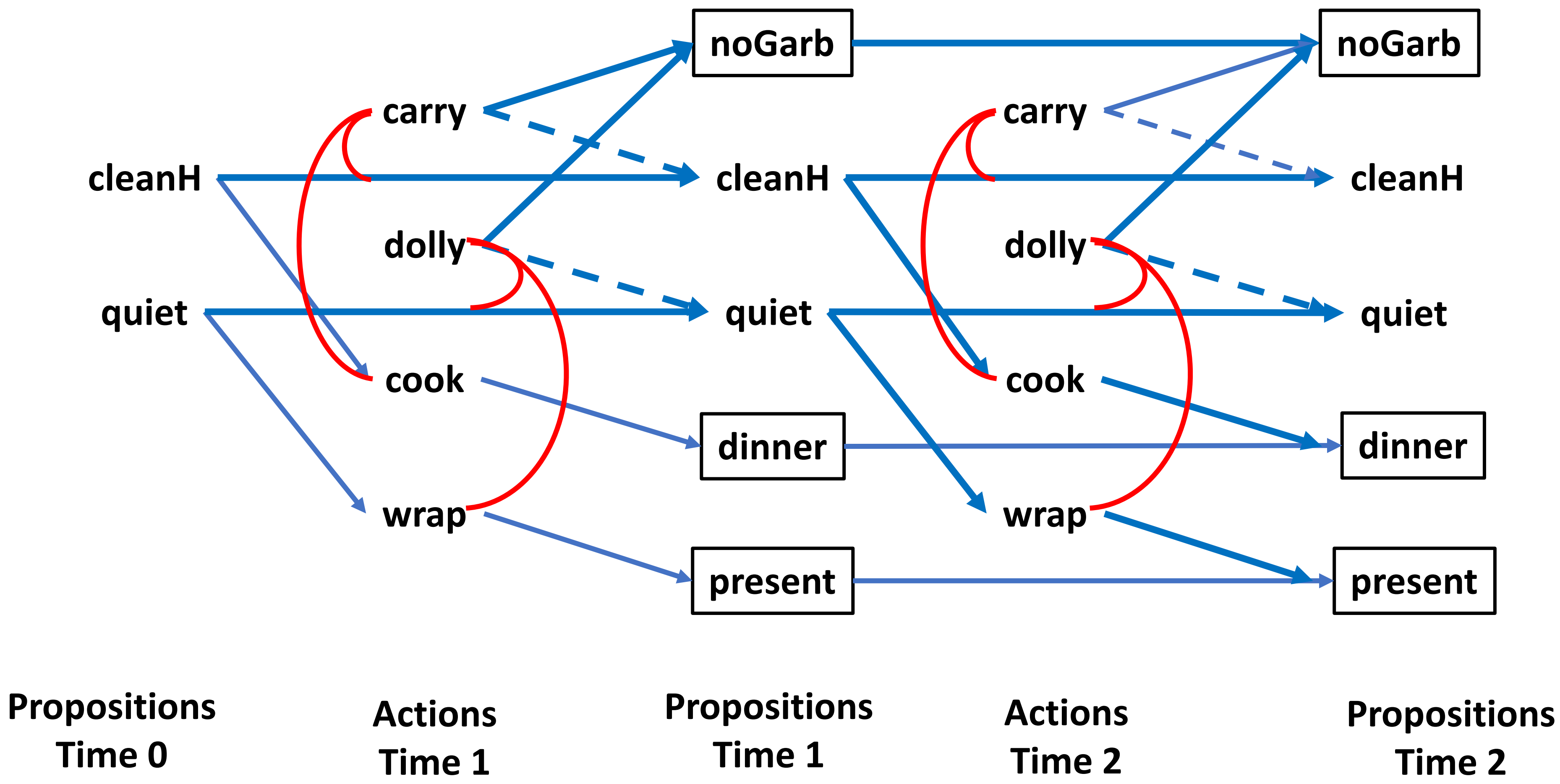
- Favor No-ops over other actions.
  - guarantees the plan will avoid redundant plan steps.

Search Action Layer 1

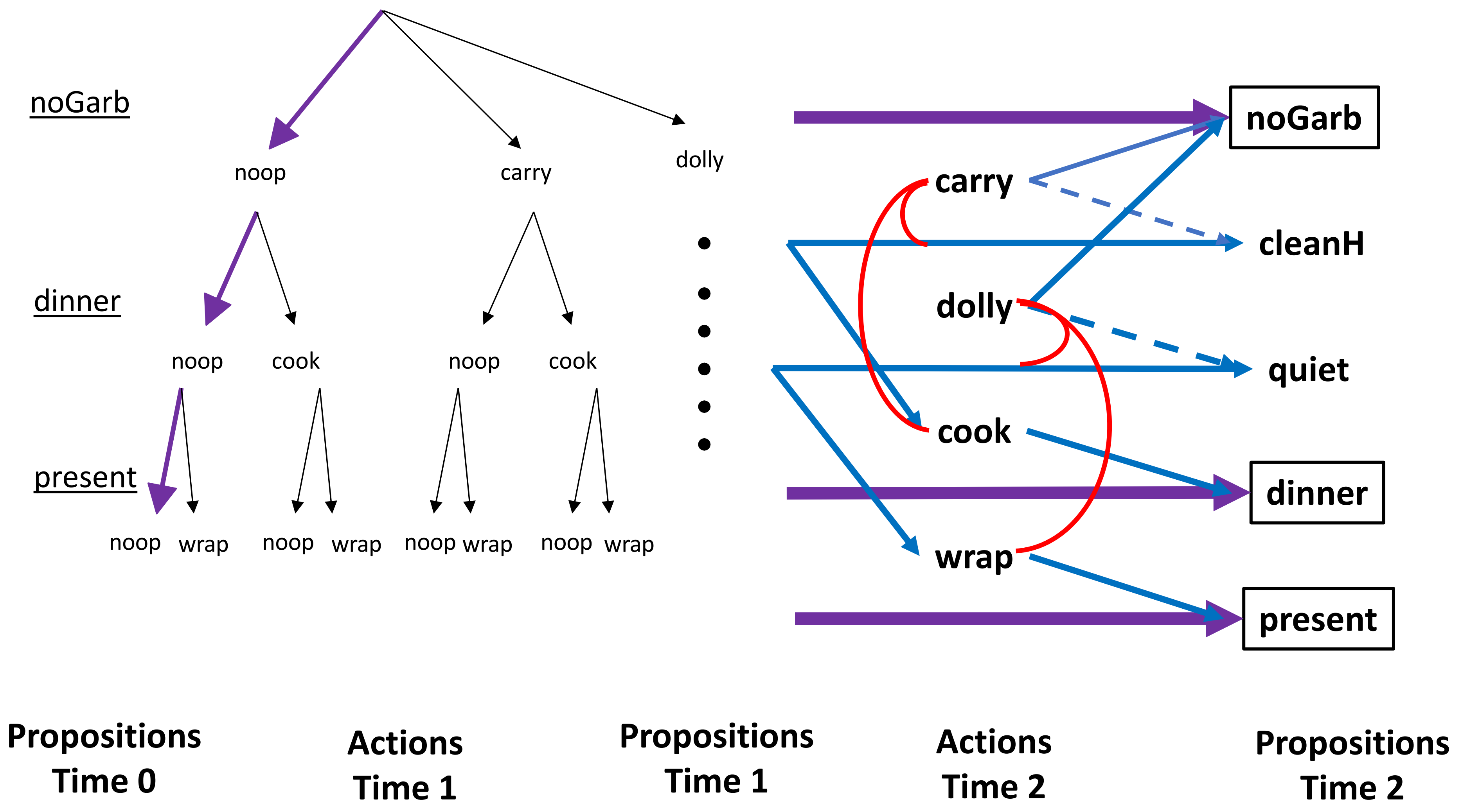




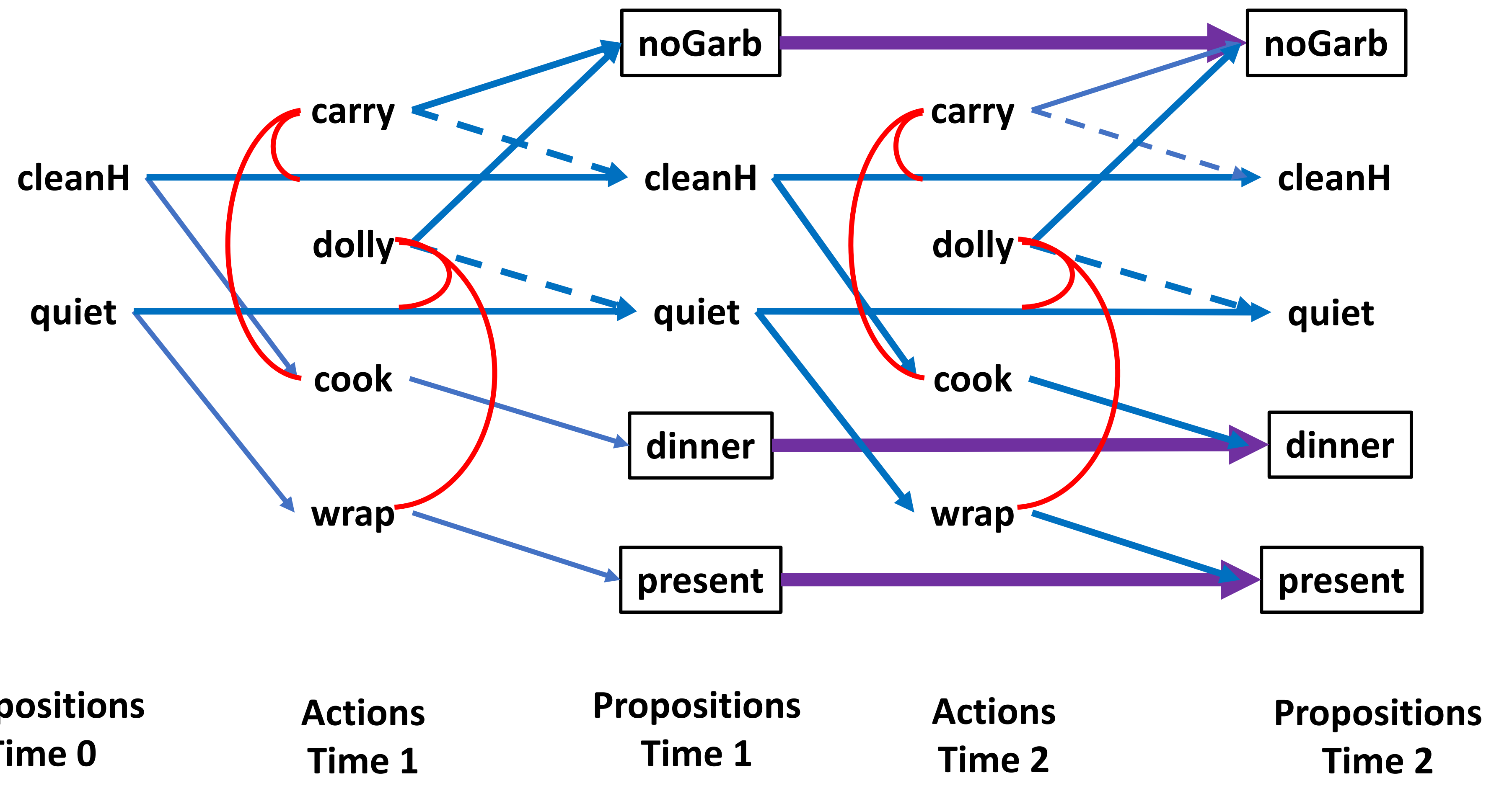
Extend & Search Action Layer 2



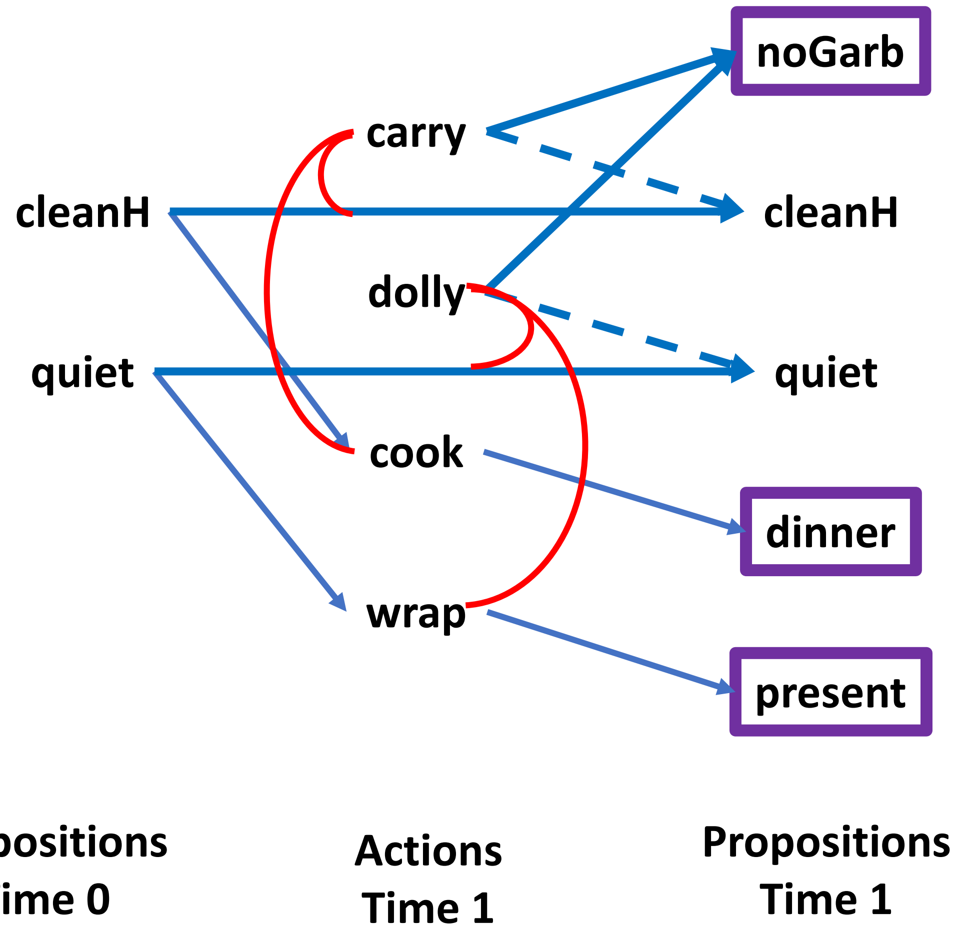
Search Action Layer 2

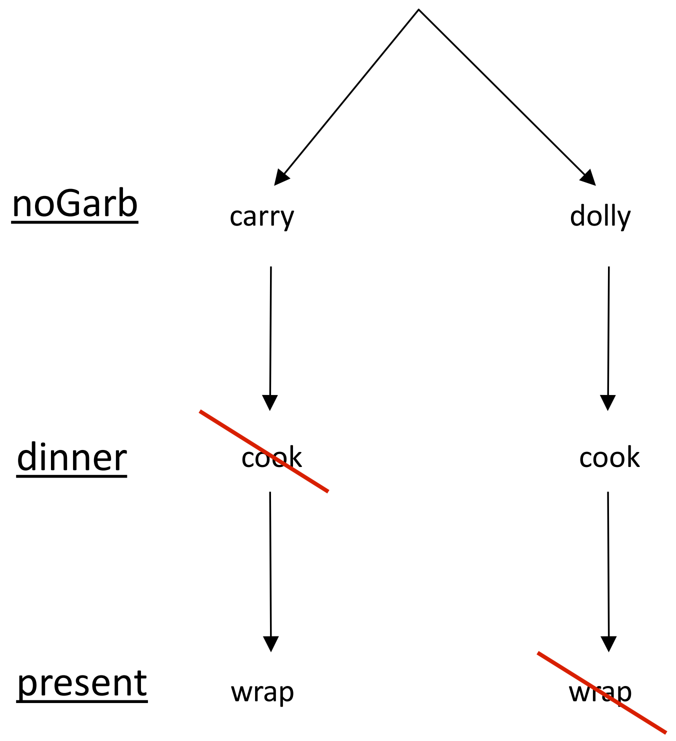


Backup and Search Action Layer 1



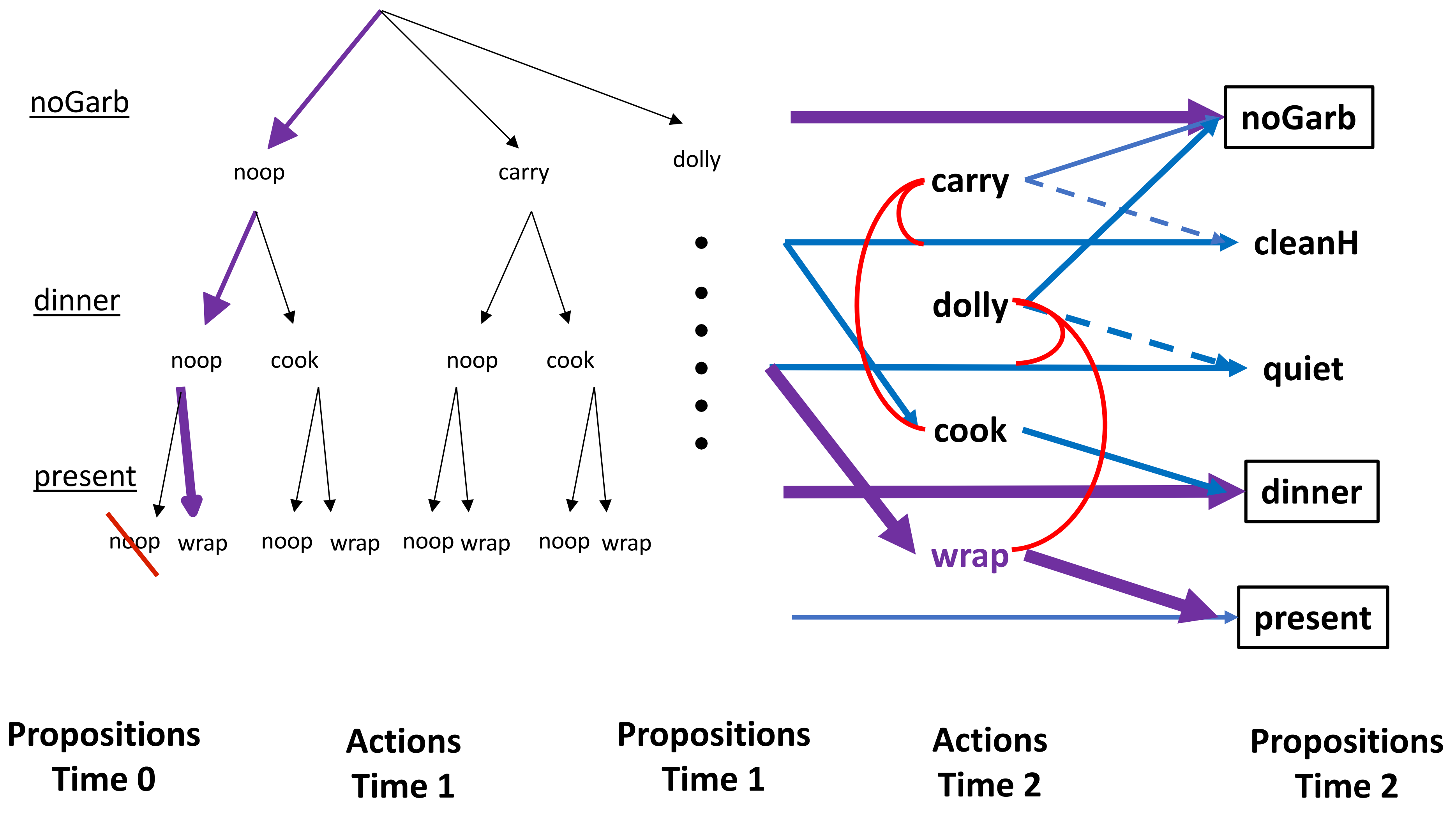
Backup and Search Action Layer 1



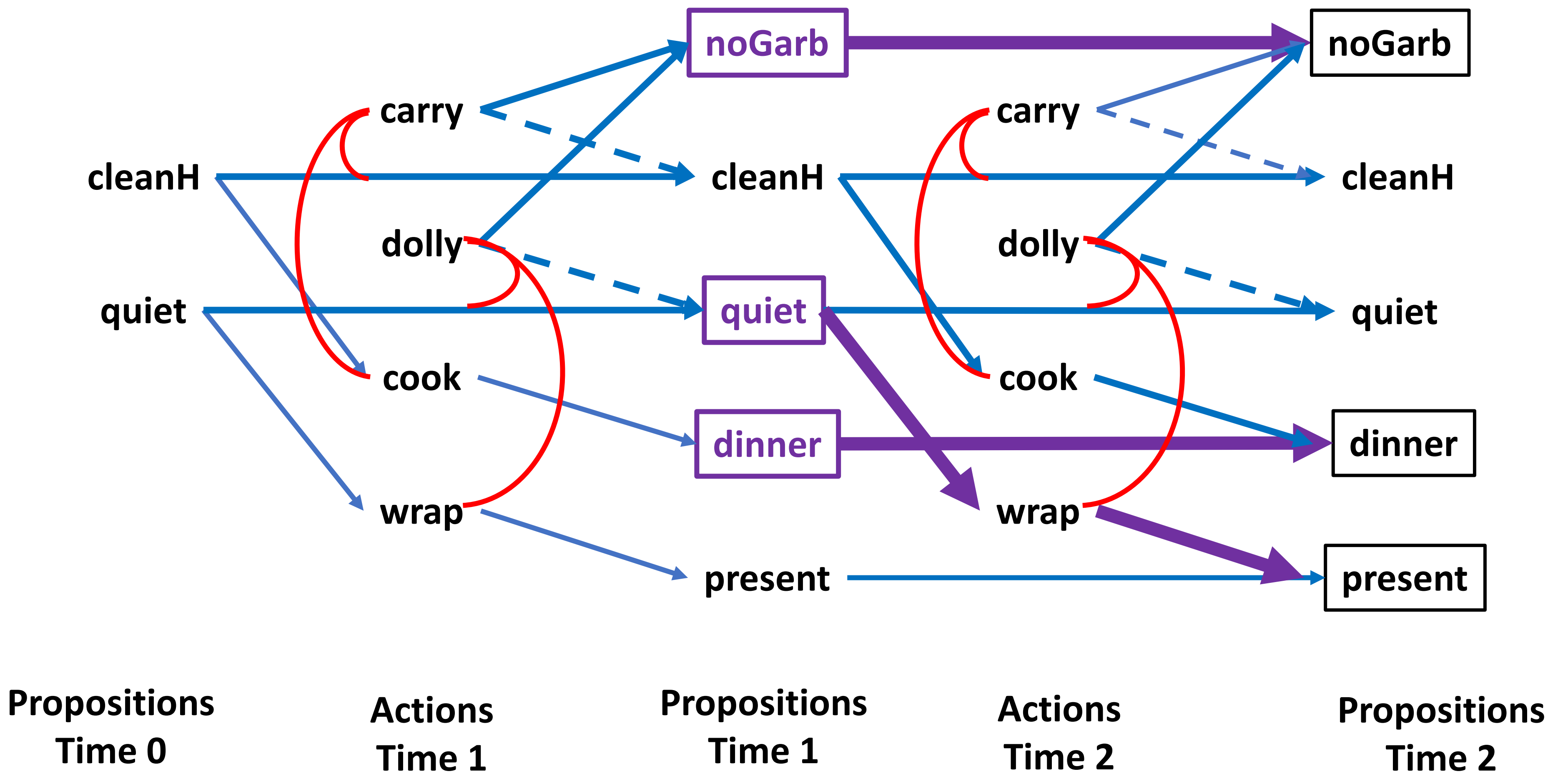


Backtrack!

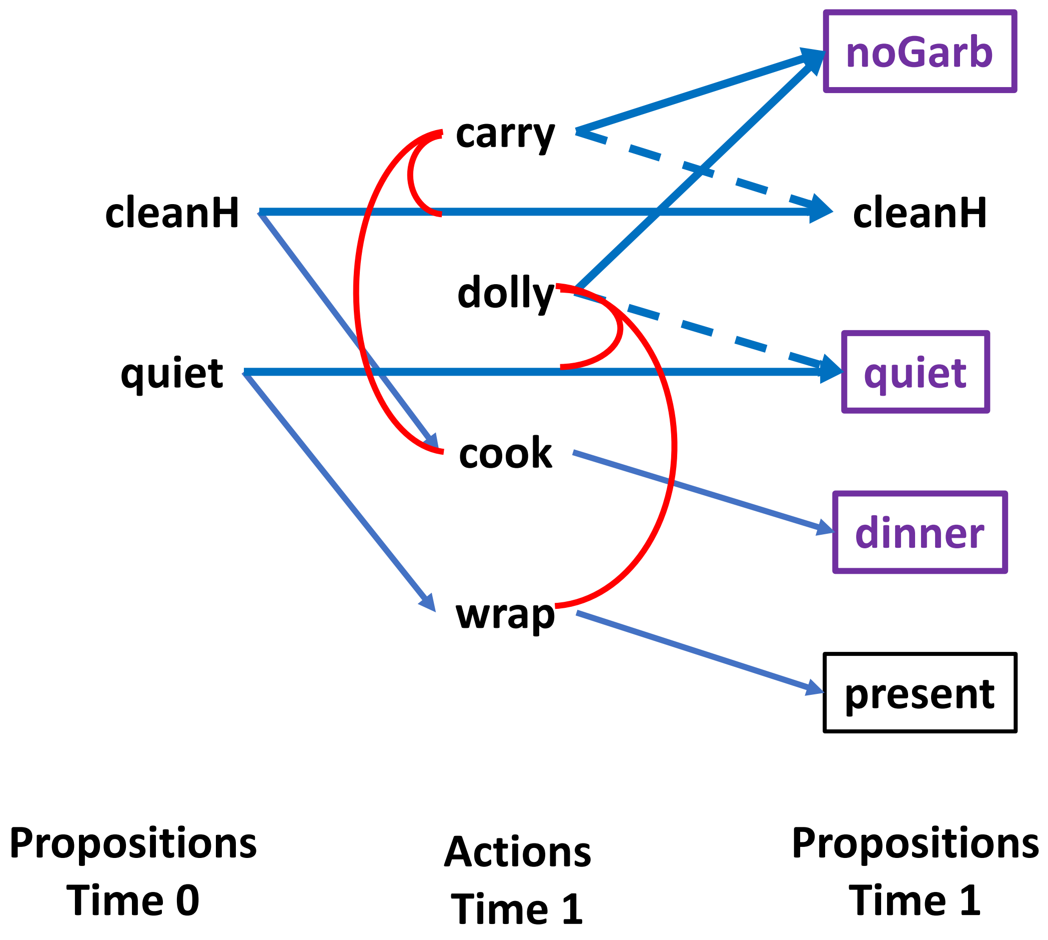
Search Action Layer 2 Again!

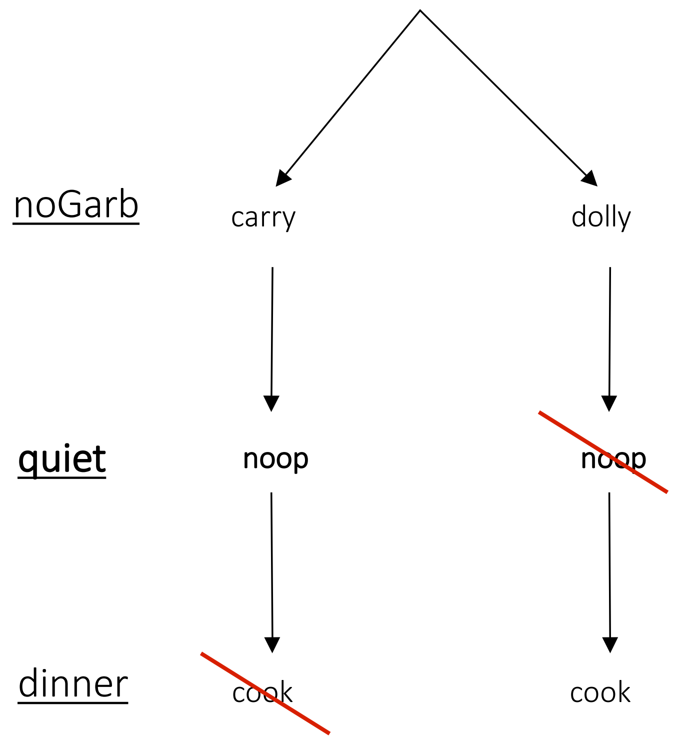


Search Action Layer 0



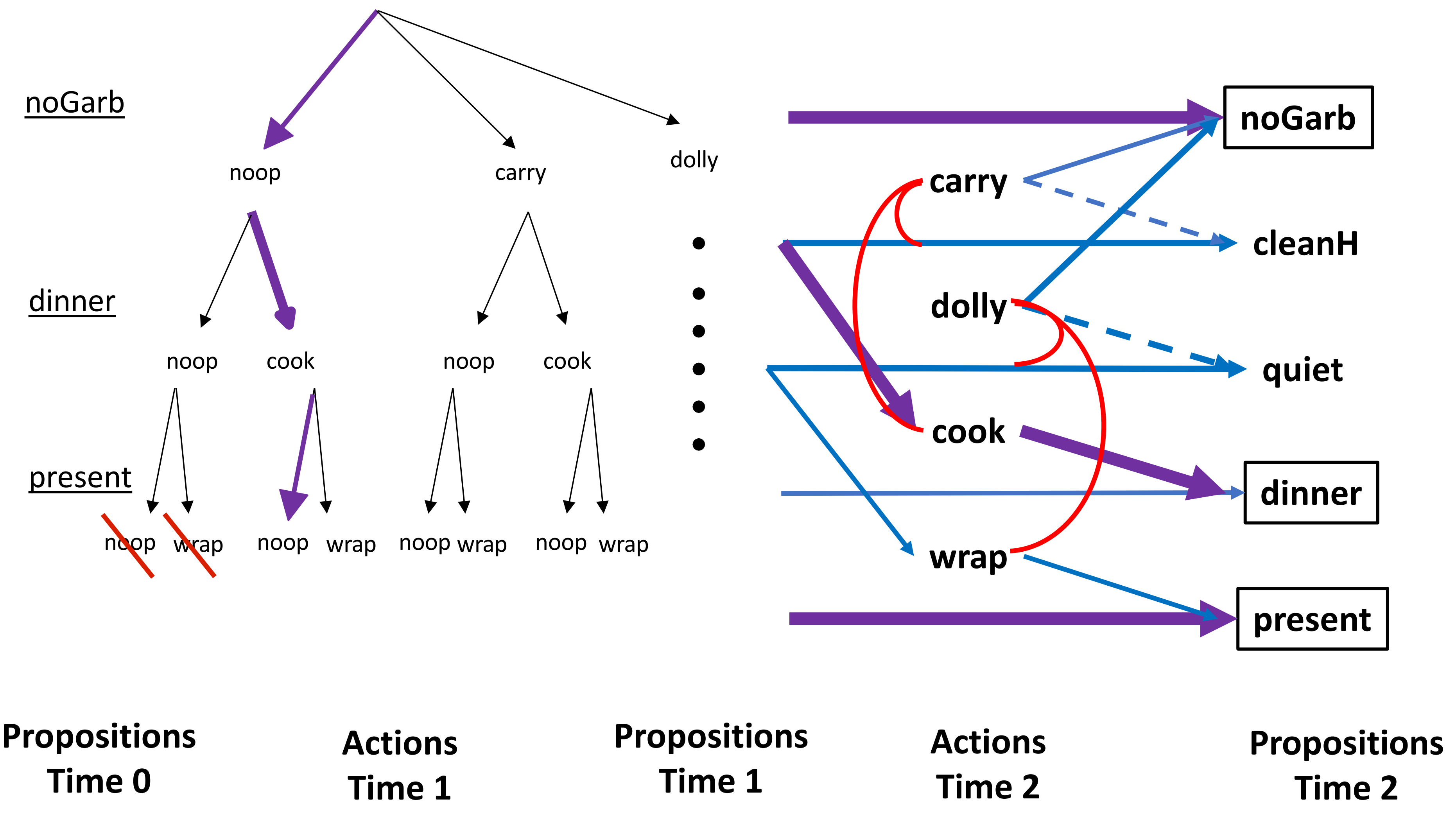
Search Action Layer 0



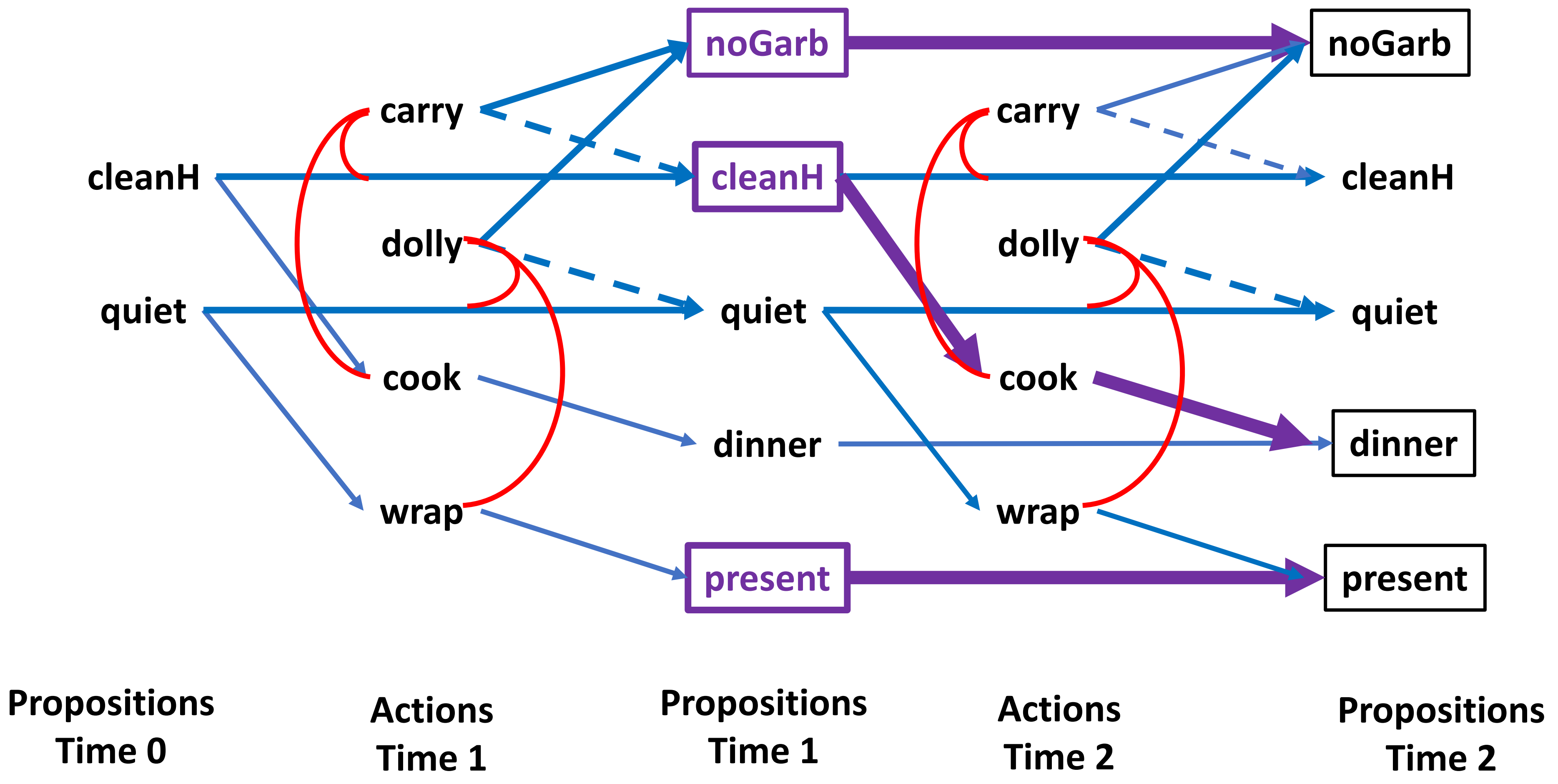


Backtrack!

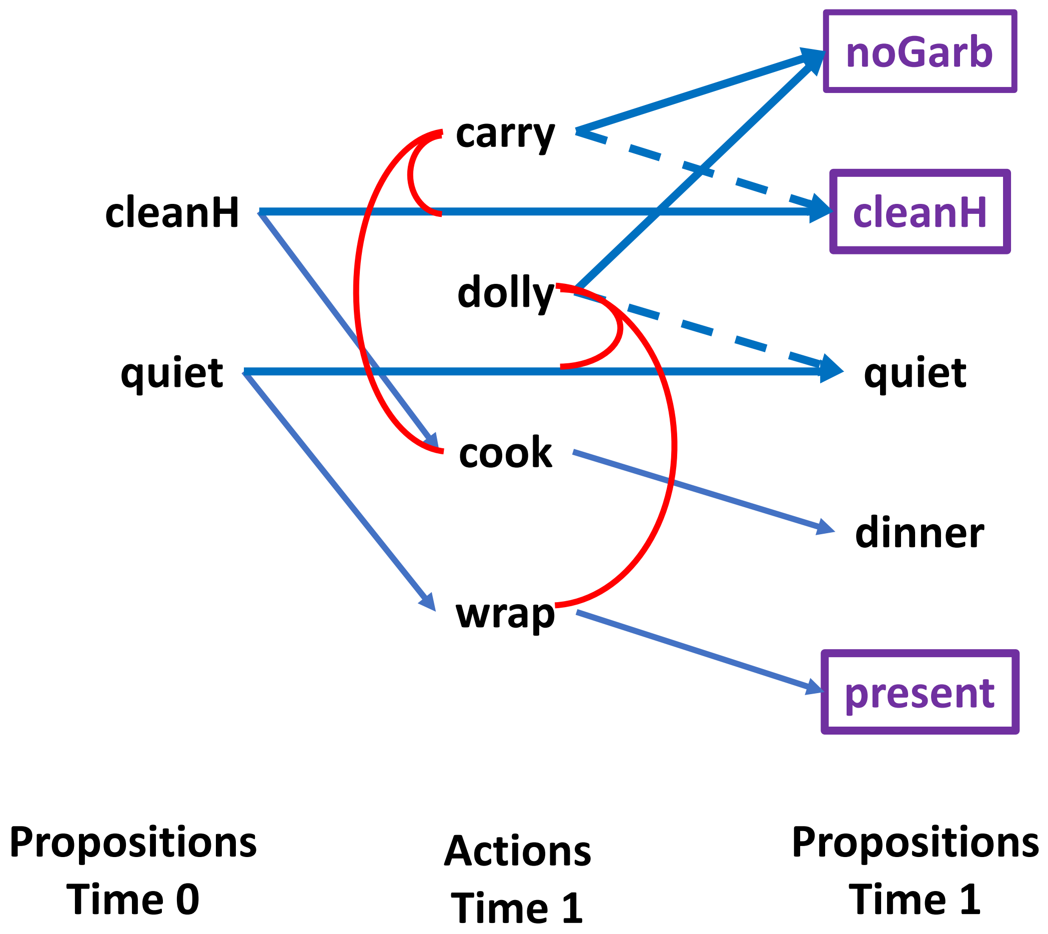
Search Action Layer 2 Again!

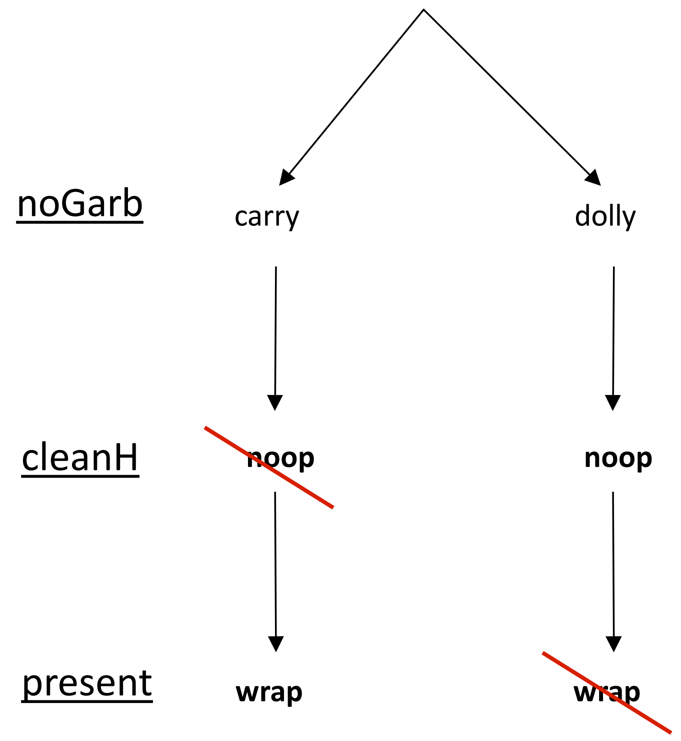


Search Action Layer 0



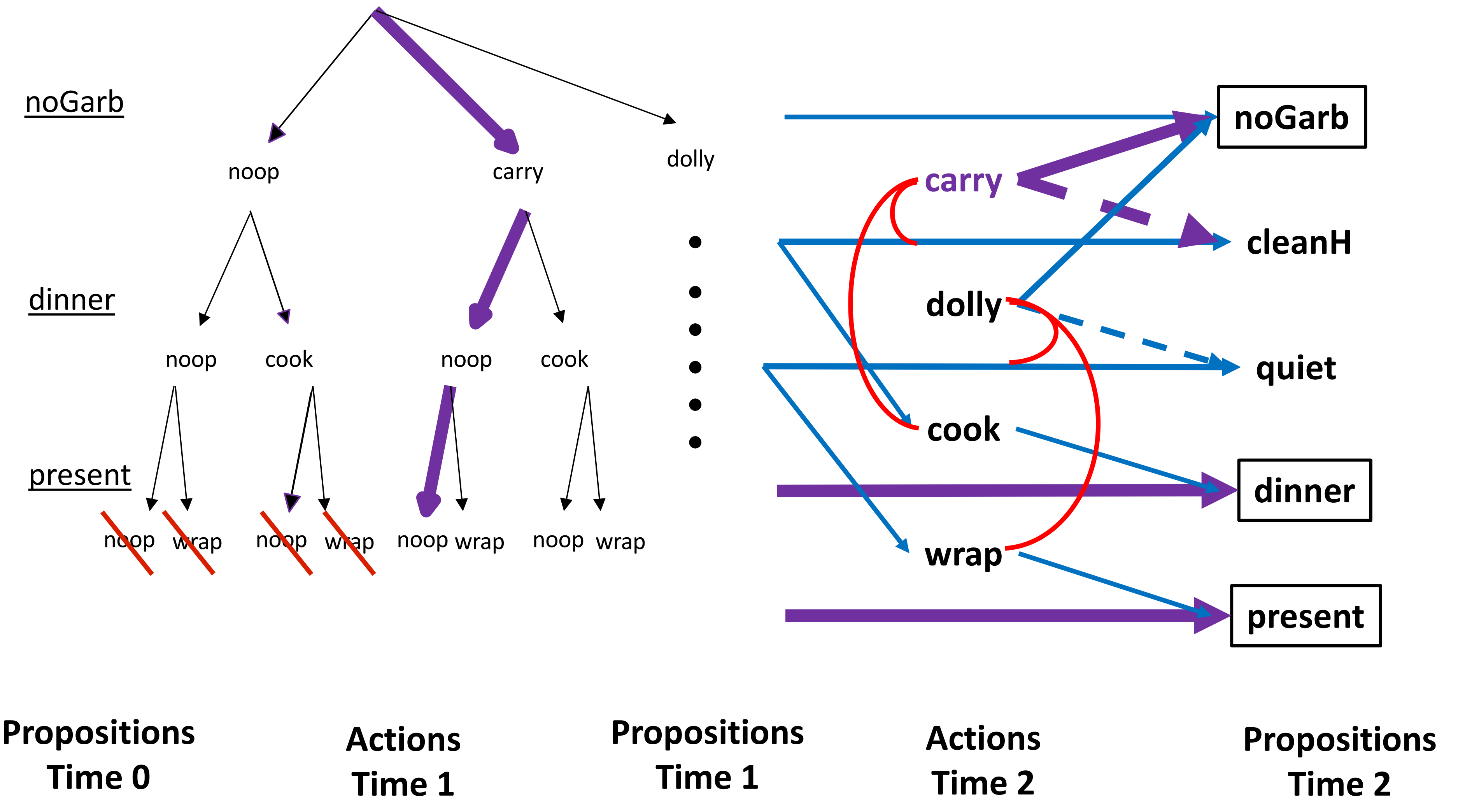
Search Action Layer 0



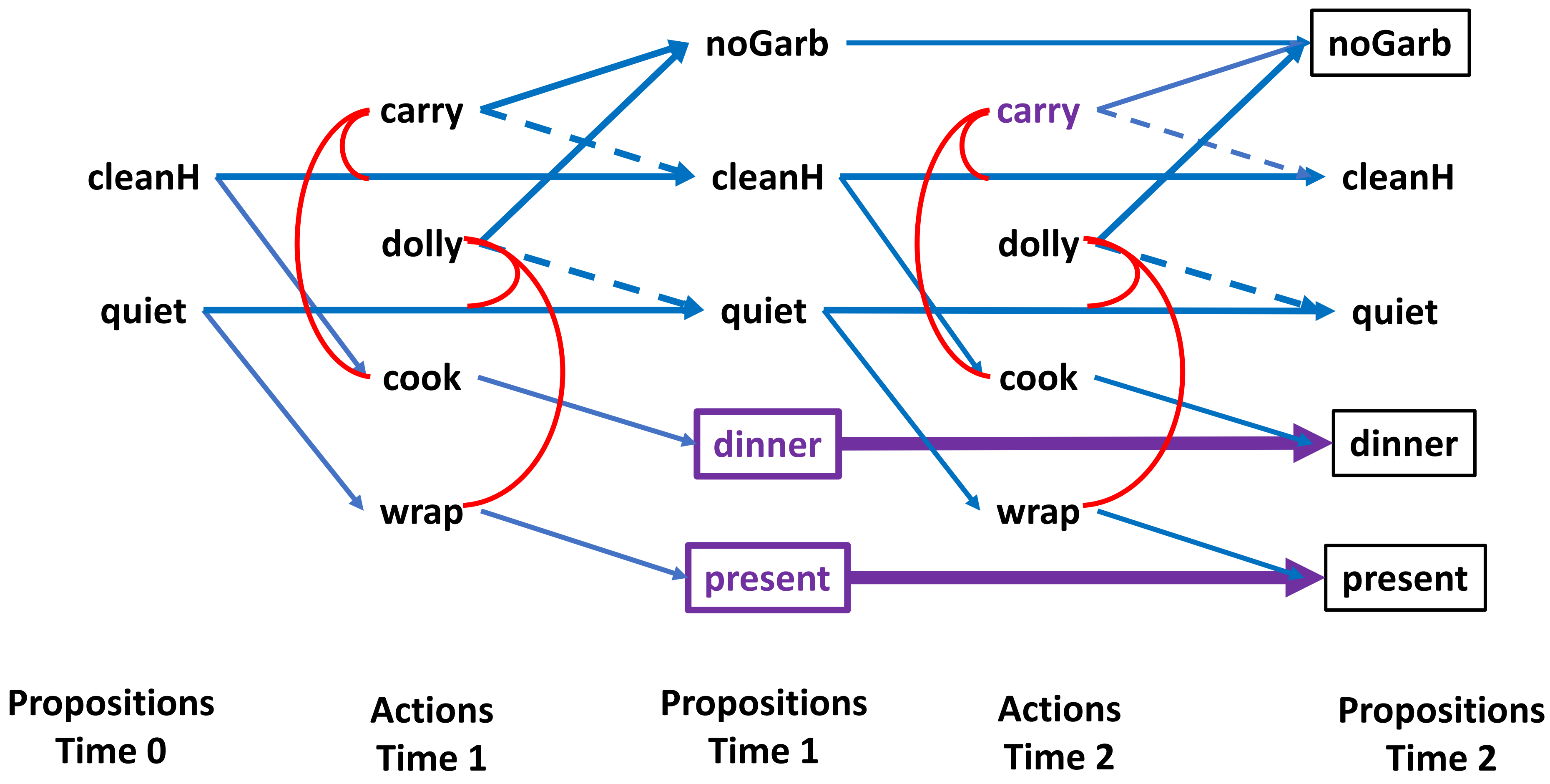


Backtrack!

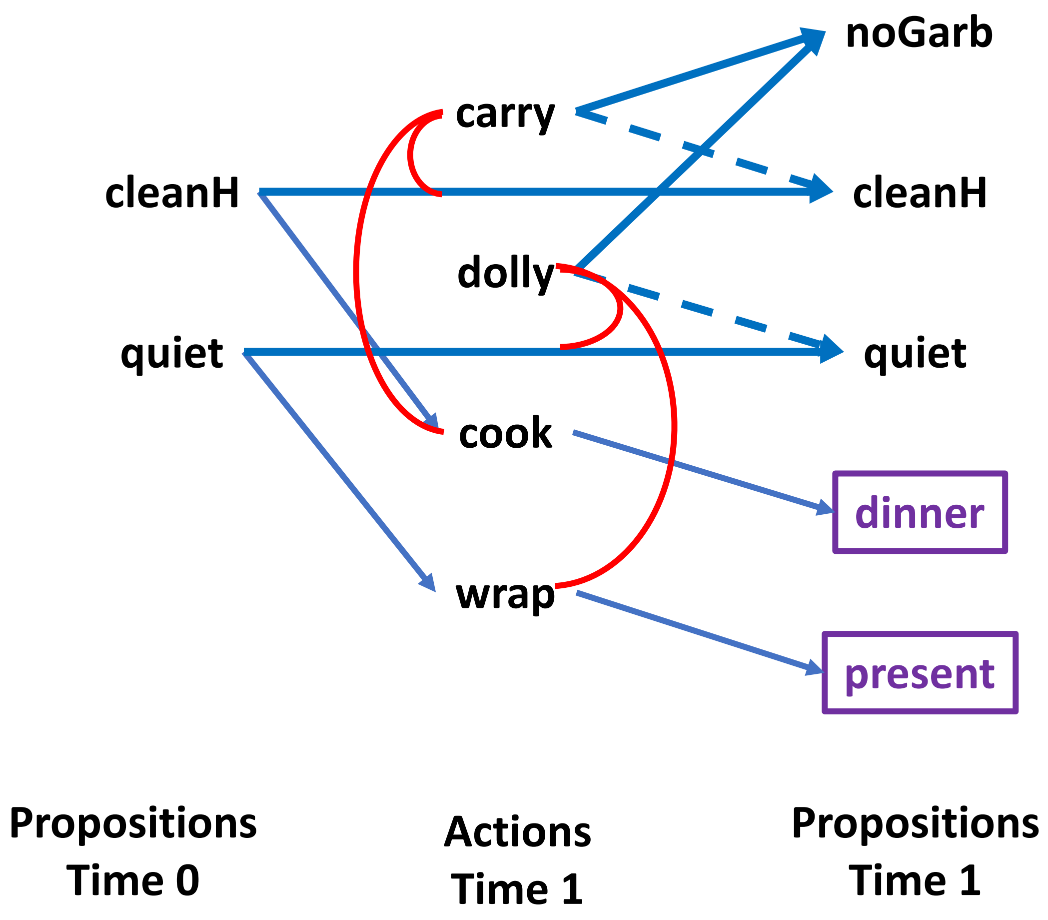
Search Action Layer 2 Again!

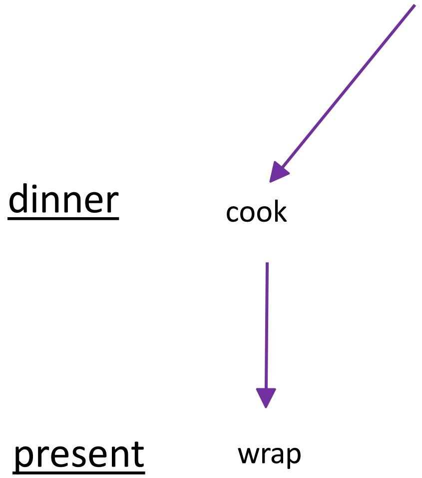


Search Action Layer 0



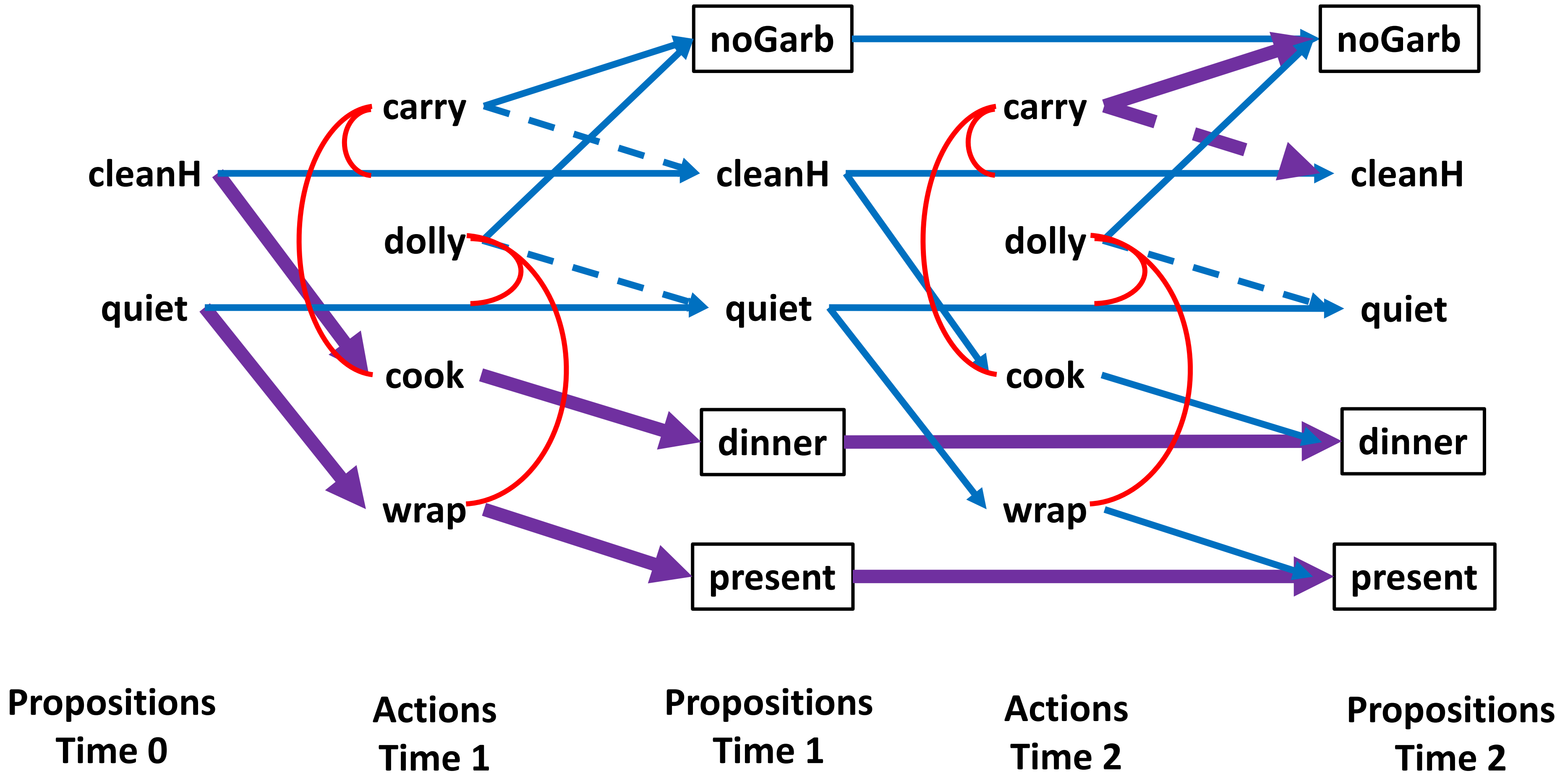
Search Action Layer 0





Consistent!

Solution: Cook & Wrap, then Carry



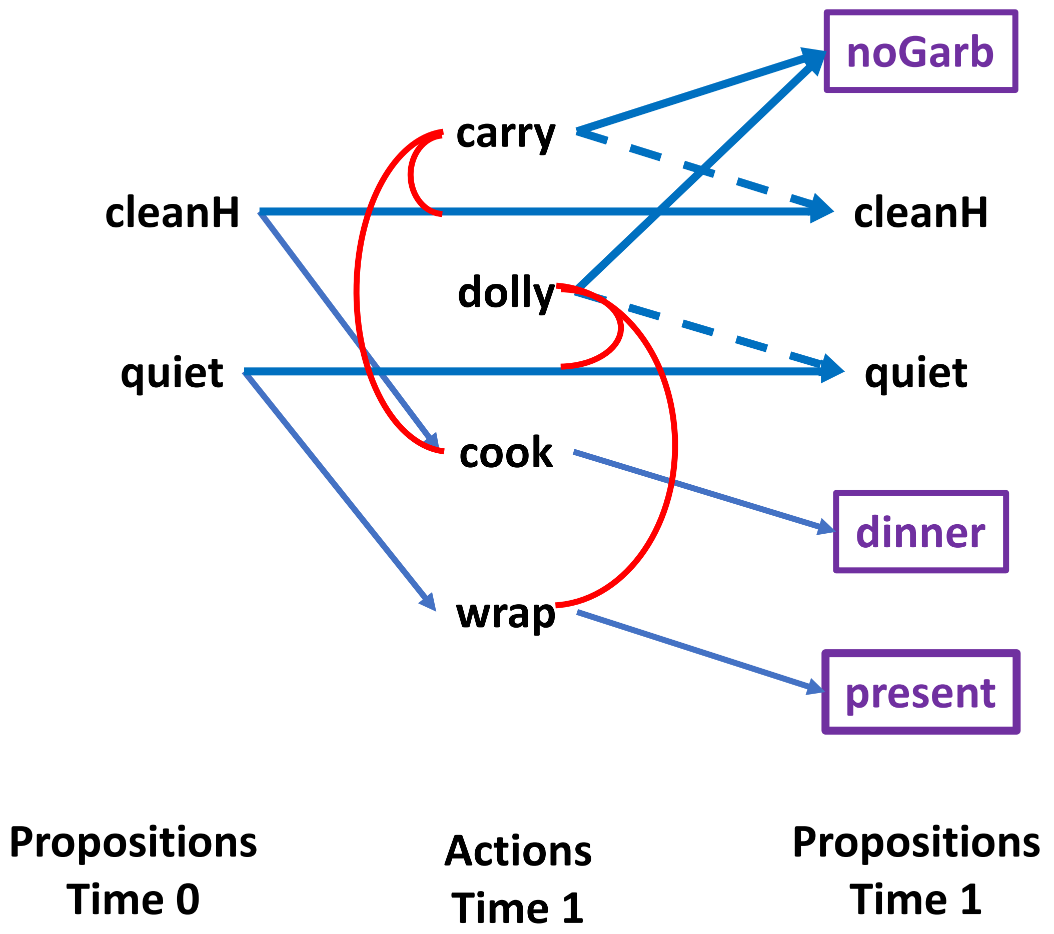
Outline

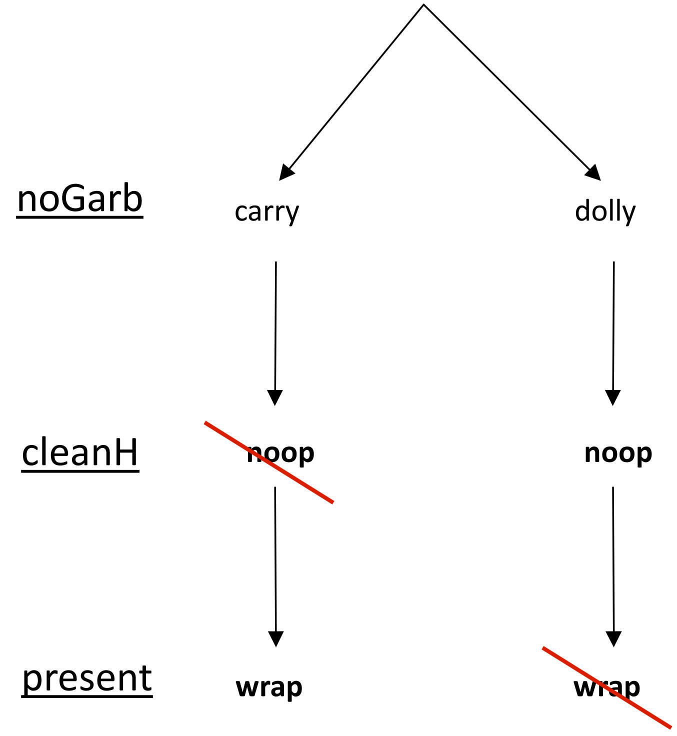
- Graph Plan
  - –Problem Statement
  - –Planning Graph Construction
  - –Plan Extraction
  - –Memos
  - –Properties
  - –Termination with Failure

Memos of Inconsistent Subgoals

- To prevent wasted search effort:
  - If a goal set at layer k cannot be achieved, then memorize the set at k
  - Check each new goal set at k against memos.
    - IF memo THEN
      - Fail
    - ELSE
      - Test by solving a CSP

Search Action Layer 0

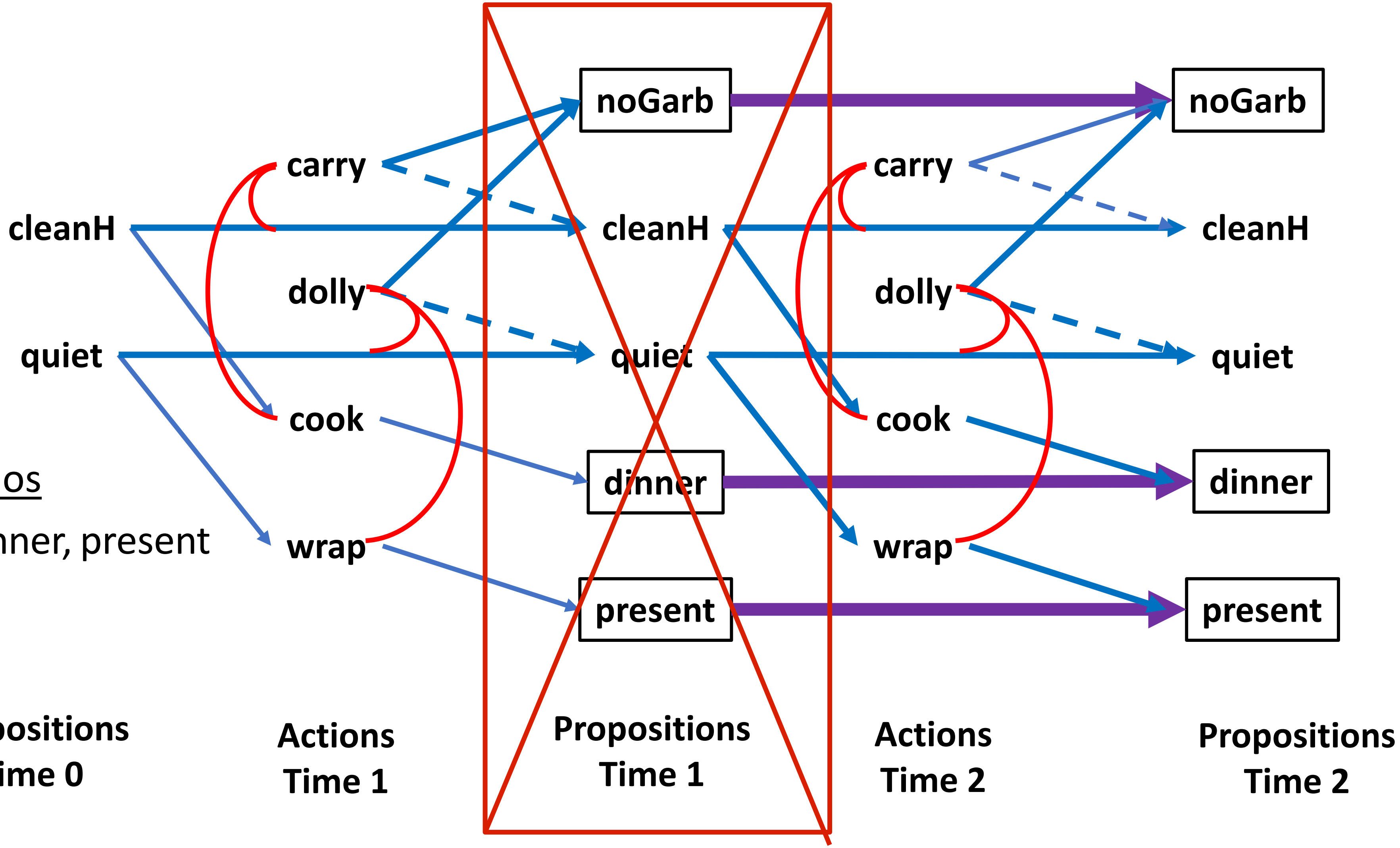




Layer 0 Memos

- noGarb, dinner, present

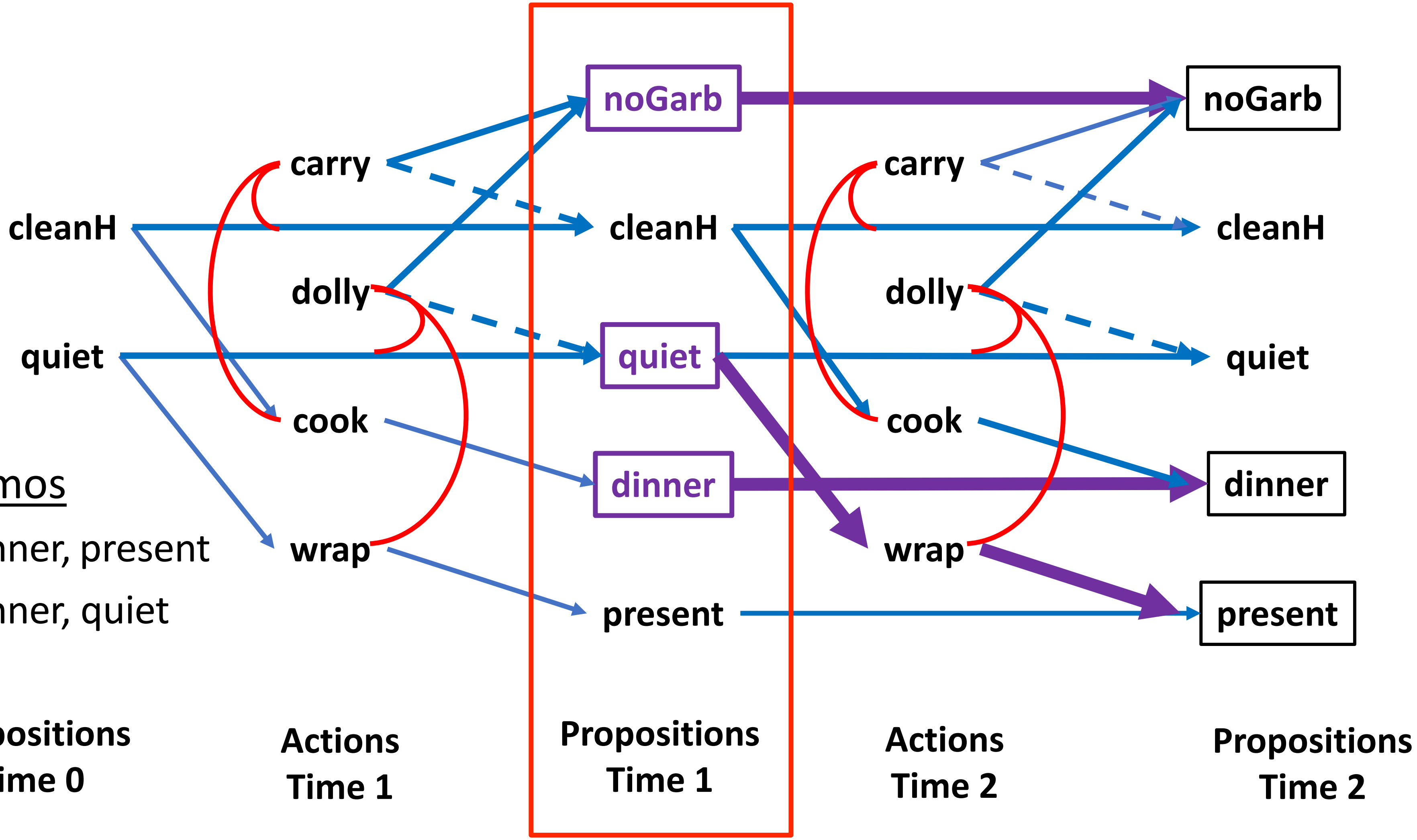
Search Layer 1: Check L0 memos



Layer 0 Memos

- noGarb, dinner, present

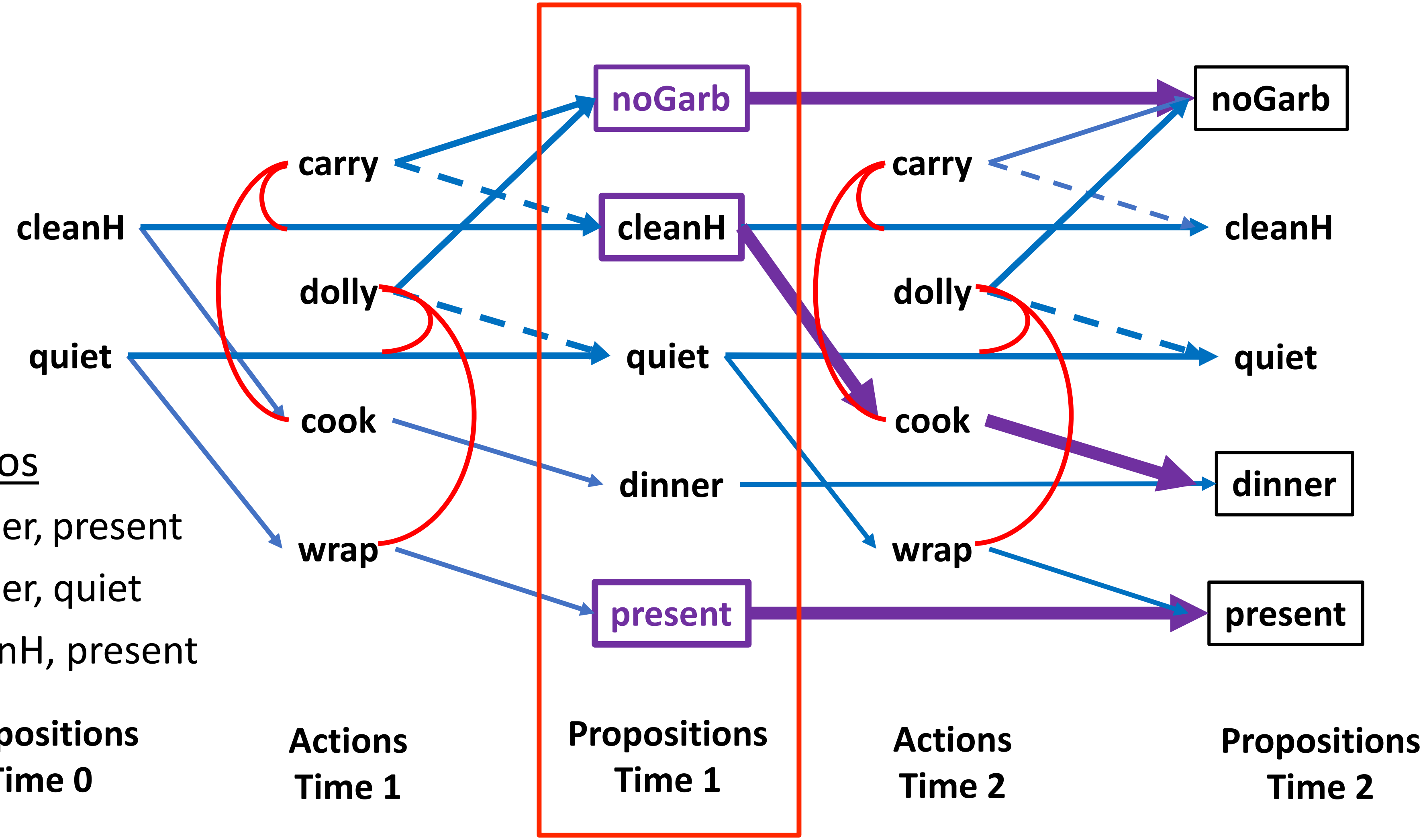
Search Layer 1: New Memo 2



Layer 0 Memos

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

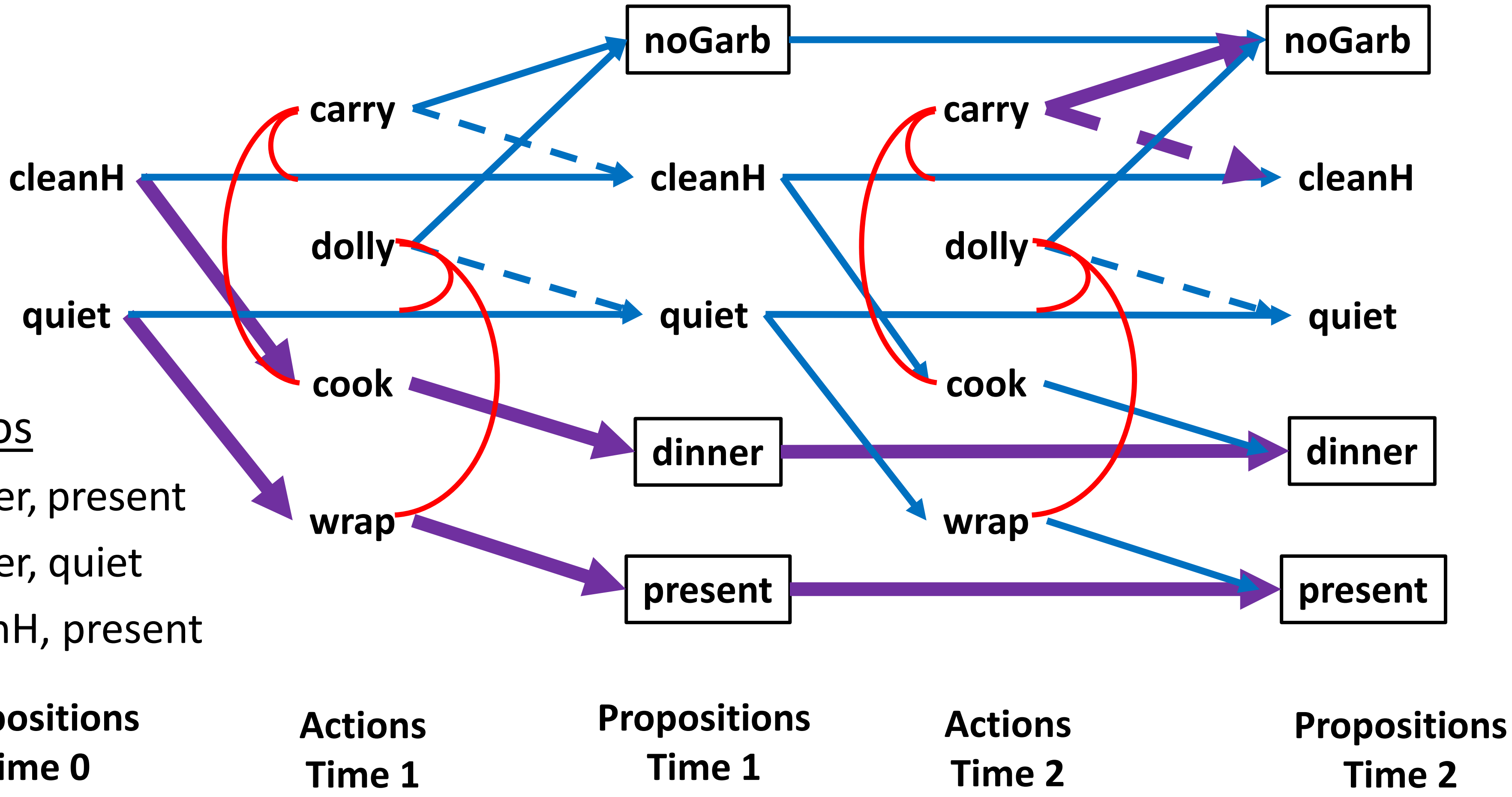
Search Layer 1: New Memo 3



Layer 0 Memos

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

Solution Found: (Not a Memo)



Layer 0 Memos

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

Outline

- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan Extraction
  - Memos
  - Properties
  - Termination with Failure

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.

→ 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

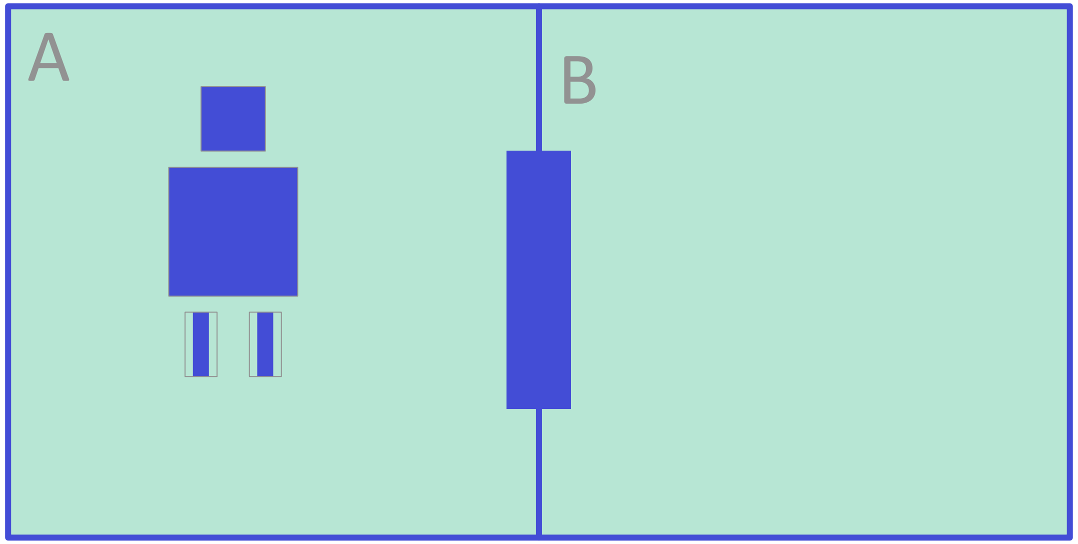
- pre: robot in ?X, door is open
- add: robot in ?Y
- del: robot in ?X

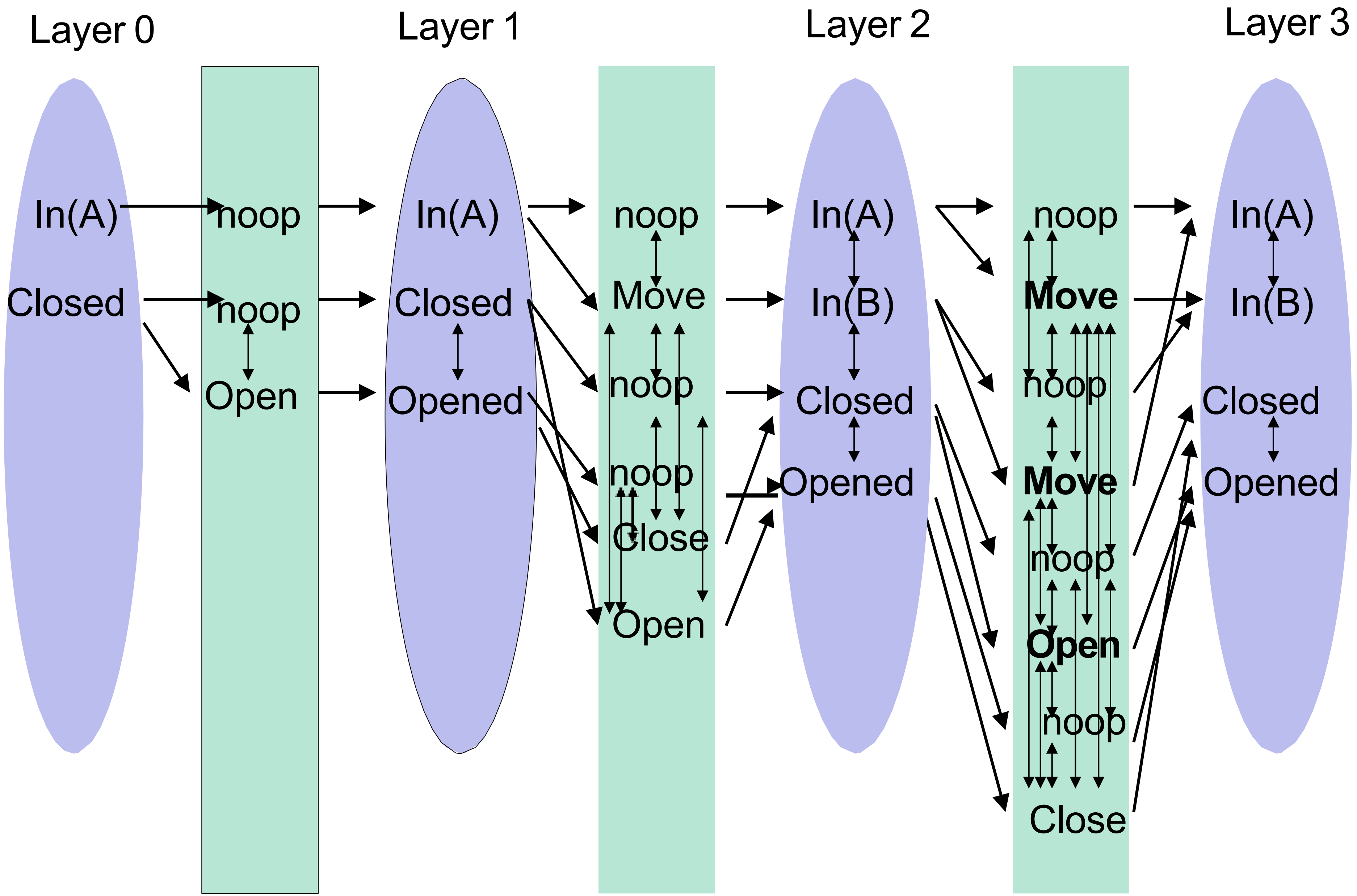
Open door

- 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?

Graph Search Properties

Move from one room to another

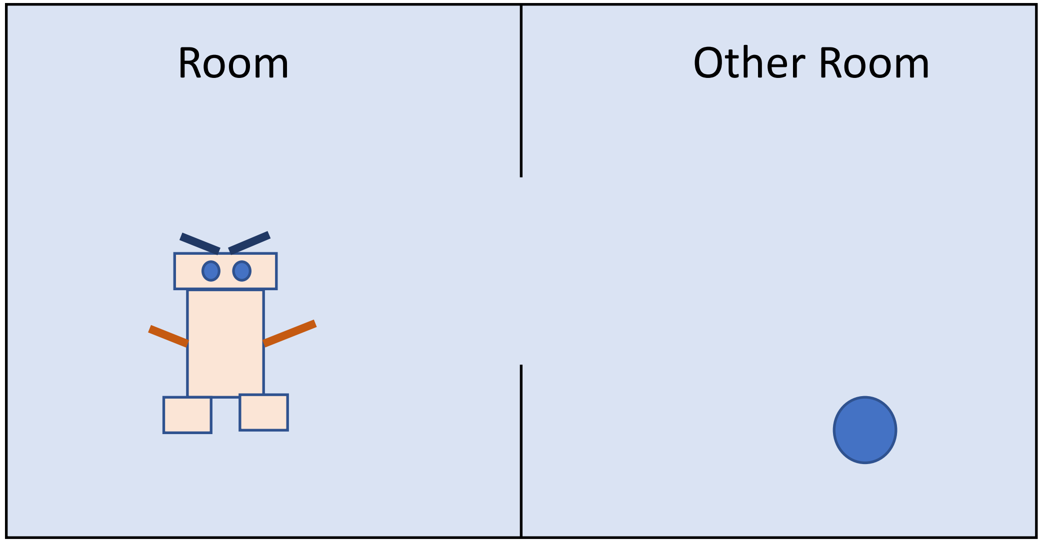
- pre: robot in first room
- add: robot in second room
- del: robot in first room

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

- Graph Plan
  - Problem Statement
  - Planning Graph Construction
  - Plan 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.

Key Take-aways

- Defining activity planning problems in a formal language
- Constructing a plan graph and searching for a solution
  - E.g., depth-first search, mutex, backtrack, memos
- Terminating search
  - “leveling out”