Graphplan

José Luis Ambite*

[* based in part on slides by Jim Blythe and Dan Weld]

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

Problem 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]**

Planning graph

- Directed, leveled graph
  - 2 types of nodes:
    - Proposition: P
    - Action: A
  - 3 types of edges (between levels)
    - Precondition: P -> A
    - Add: A -> P
    - Delete: A -> P
- Proposition and action levels alternate
- Action level includes actions whose preconditions are satisfied in previous level plus no-op actions (to solve frame problem).

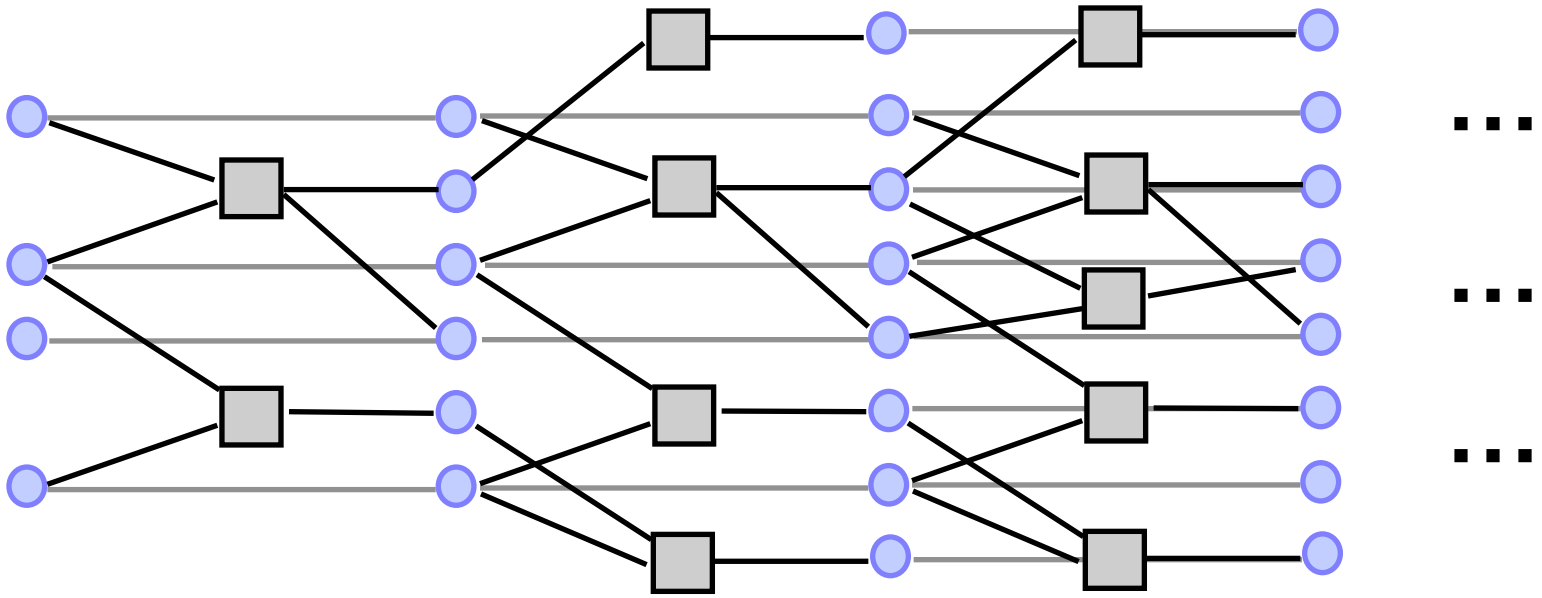
Rocket domain

```
(define (operator move)
    :parameters ((rocket ?r) (place ?from) (place ?to))
    :precondition (:and (:neq ?from ?to) (at ?r ?from) (has-fuel ?r))
    :effect (:and (at ?r ?to) (:not (at ?r ?from)) (:not (has-fuel ?r))))

(define (operator unload)
    :parameters ((rocket ?r) (place ?p) (cargo ?c))
    :precondition (:and (at ?r ?p) (in ?c ?r))
    :effect (:and (:not (in ?c ?r)) (at ?c ?p)))

(define (operator load)
    :parameters ((rocket ?r) (place ?p) (cargo ?c))
    :precondition (:and (at ?r ?p) (at ?c ?p))
    :effect (:and (:not (at ?c ?p)) (in ?c ?r)))
```

Planning graph



Constructing the planning graph

- Level P_1: all literals from the initial state
- Add an action in level A_i if all its preconditions are present in level P_i
- Add a precondition in level P_i if it is the effect of some action in level A_{i-1} (including no-ops)
- Maintain a set of exclusion relations to eliminate incompatible propositions and actions (thus reducing the graph size)

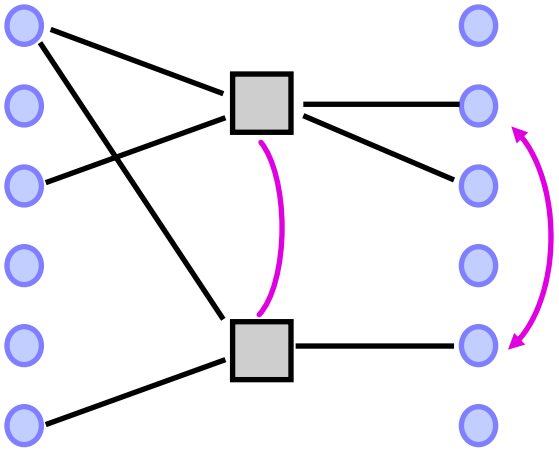
$$\mathbf{P_1\ A_1\ P_2\ A_2\ \ldots\ P_{n-1}\ A_{n-1}\ P_n}$$

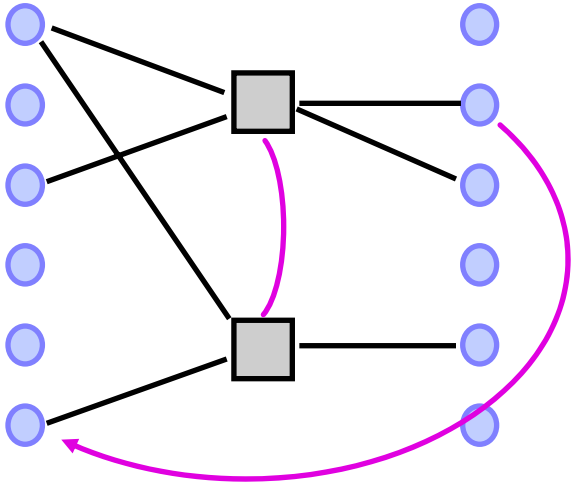
Mutual Exclusion relations

- Two actions (or literals) are mutually exclusive (mutex) at some stage if no valid plan could contain both.
- Two actions are mutex if:
  - Interference: one clobbers others' effect or precondition
  - Competing needs: mutex preconditions
- Two propositions are mutex if:
  - All ways of achieving them are mutex

Mutual Exclusion relations

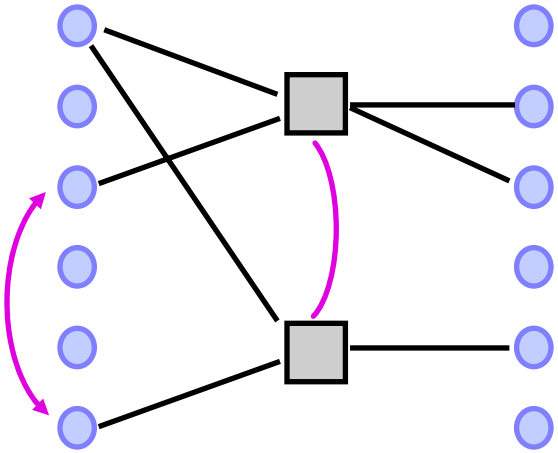
Inconsistent Effects

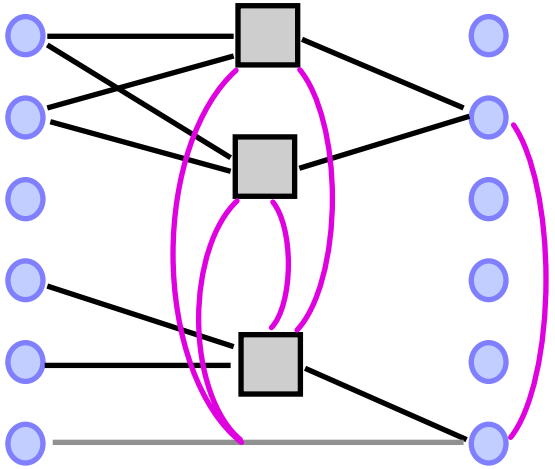




Interference (prec-effect)

Competing Needs



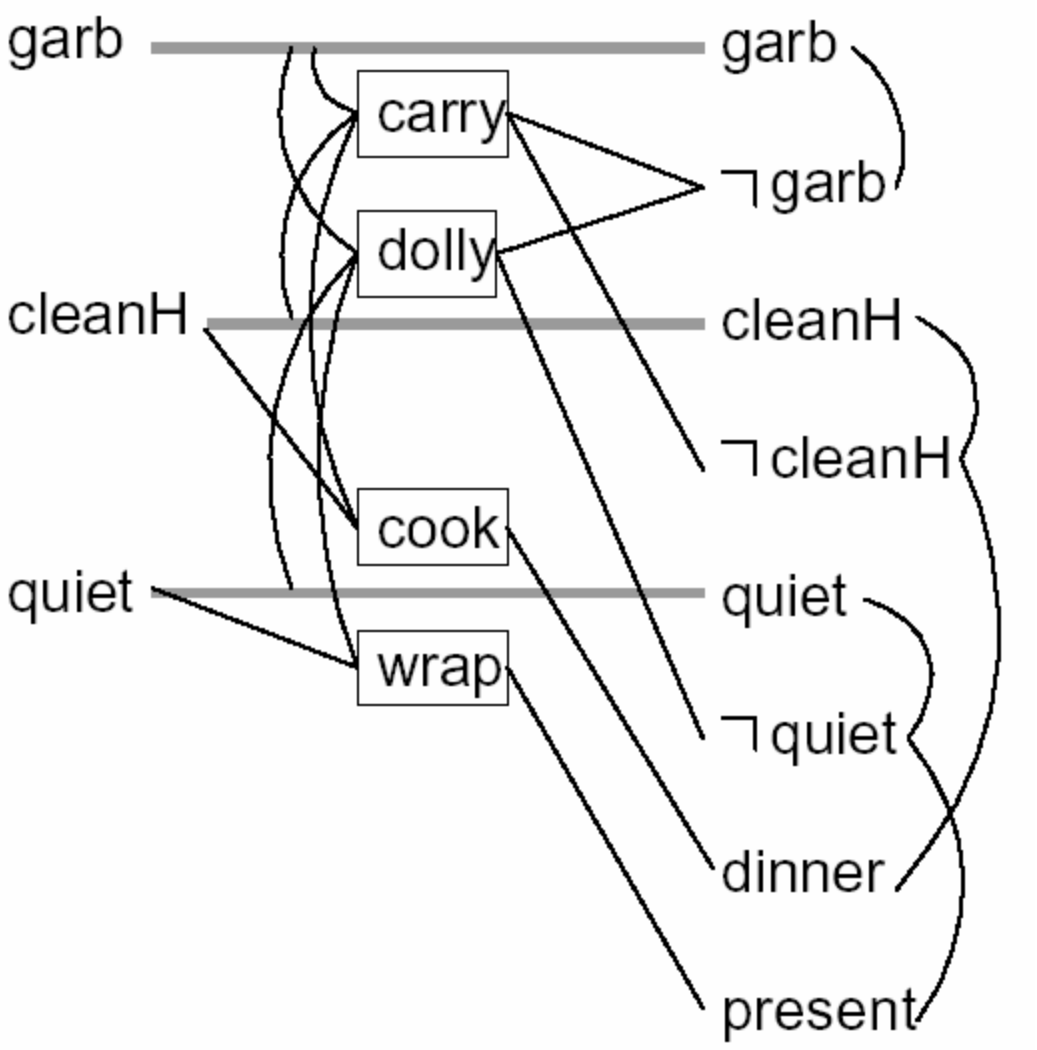


Inconsistent Support

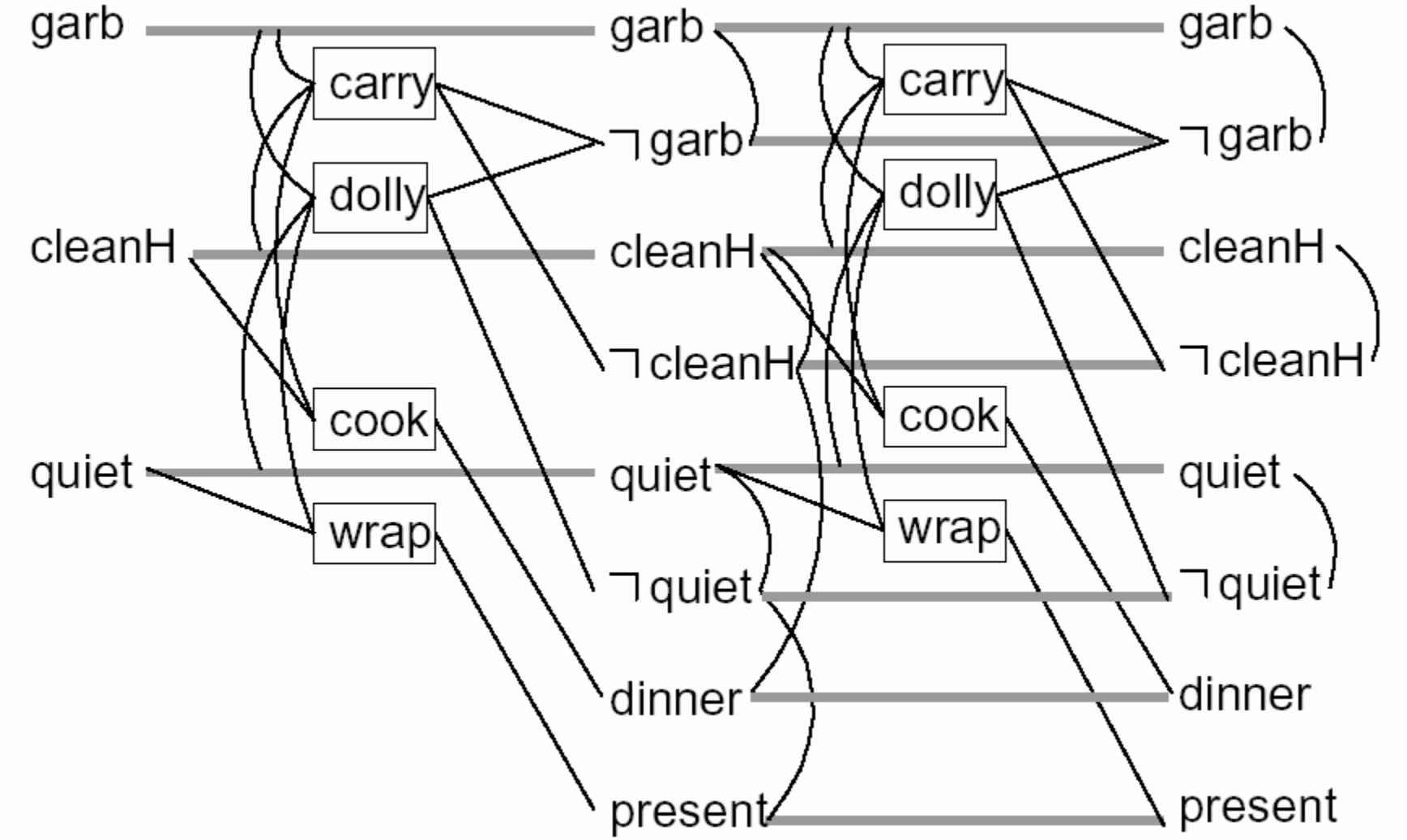
Dinner Date example

- Initial Conditions: (and (garbage) (cleanHands) (quiet))
- Goal: (and (dinner) (present) (not (garbage))
- Actions:
  - Cook :precondition (cleanHands)
    :effect (dinner)
  - Wrap :precondition (quiet)
    :effect (present)
  - Carry :precondition
    :effect (and (not (garbage)) (not (cleanHands))
  - Dolly :precondition
    :effect (and (not (garbage)) (not (quiet)))

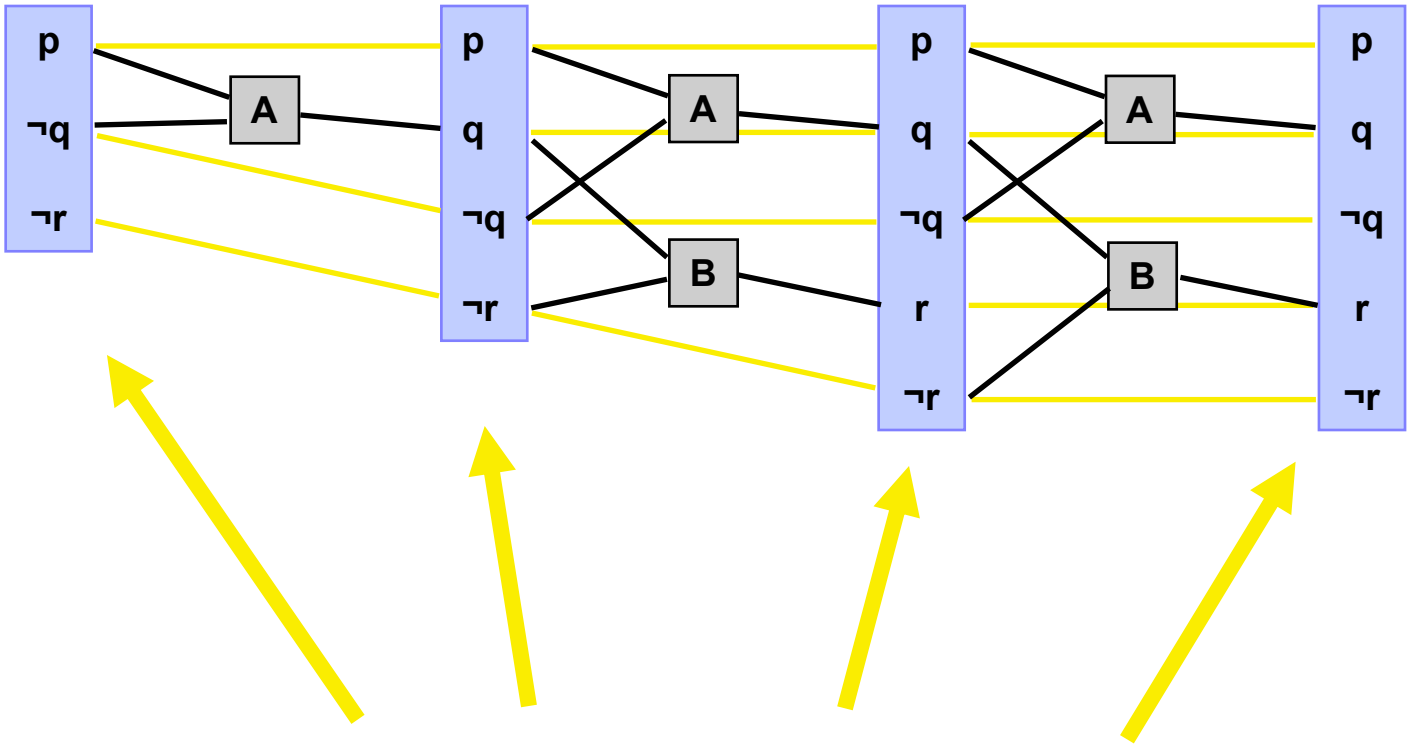
Dinner Date example



Dinner Date example



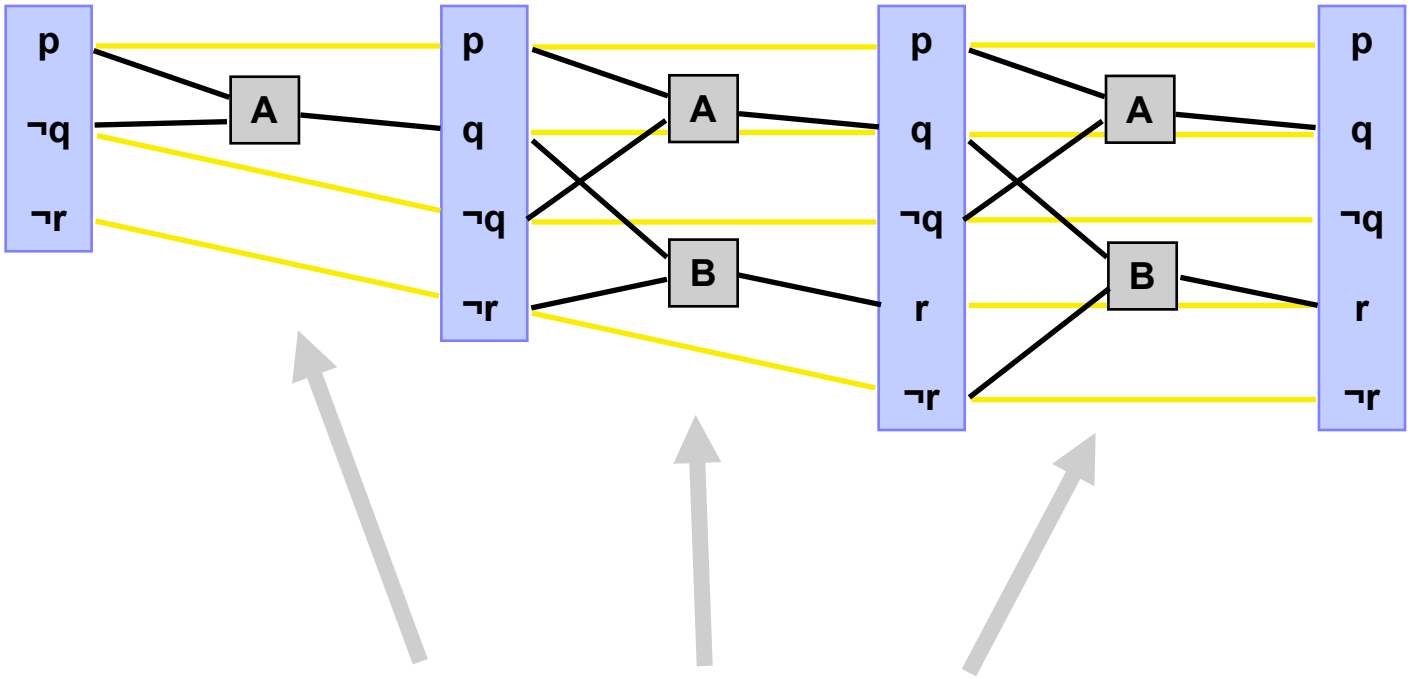
Observation 1



Propositions monotonically increase

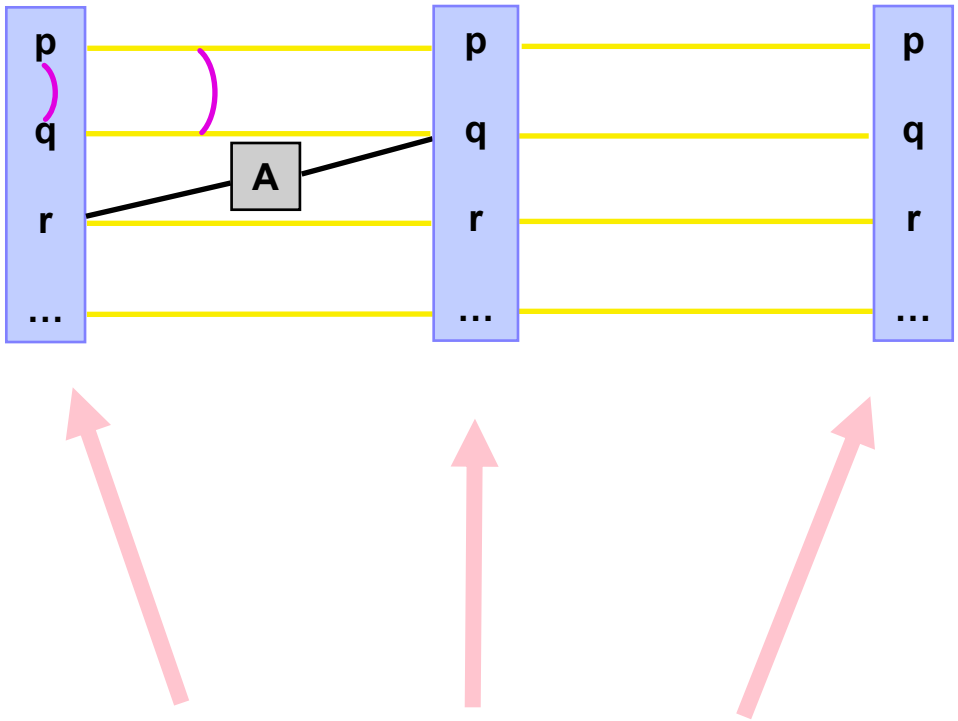
(always carried forward by no-ops)

Observation 2



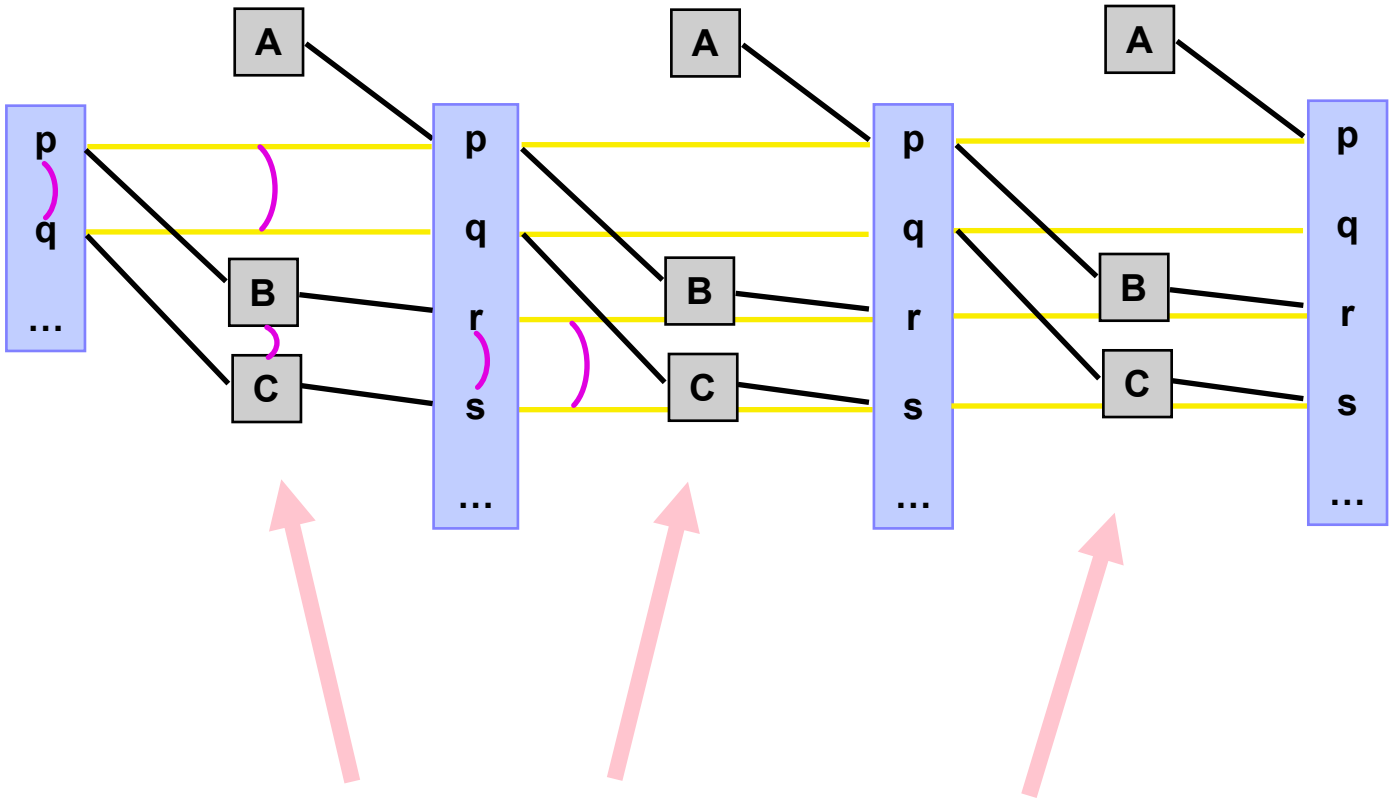
Actions monotonically increase

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.

Valid plan

A valid plan is a planning graph where:

- Actions at the same level don't interfere
- Each action's preconditions are made true by the plan
- Goals are satisfied

GraphPlan algorithm

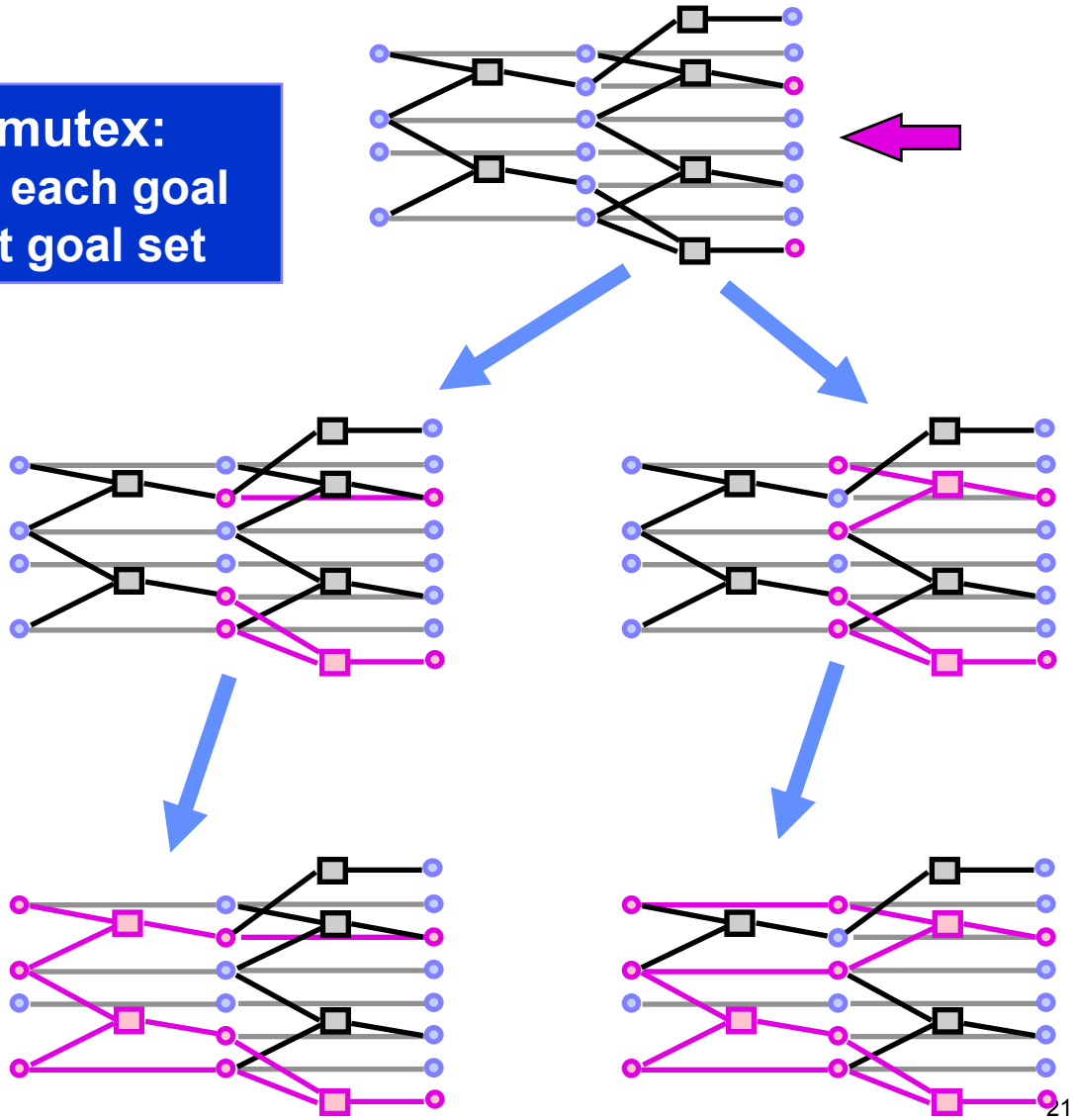
- Grow the planning graph (PG) until all goals are reachable and not mutex. (If PG levels off first, fail)
- Search the PG for a valid plan
- If non found, add a level to the PG and try again

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)

Plan Graph Search

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



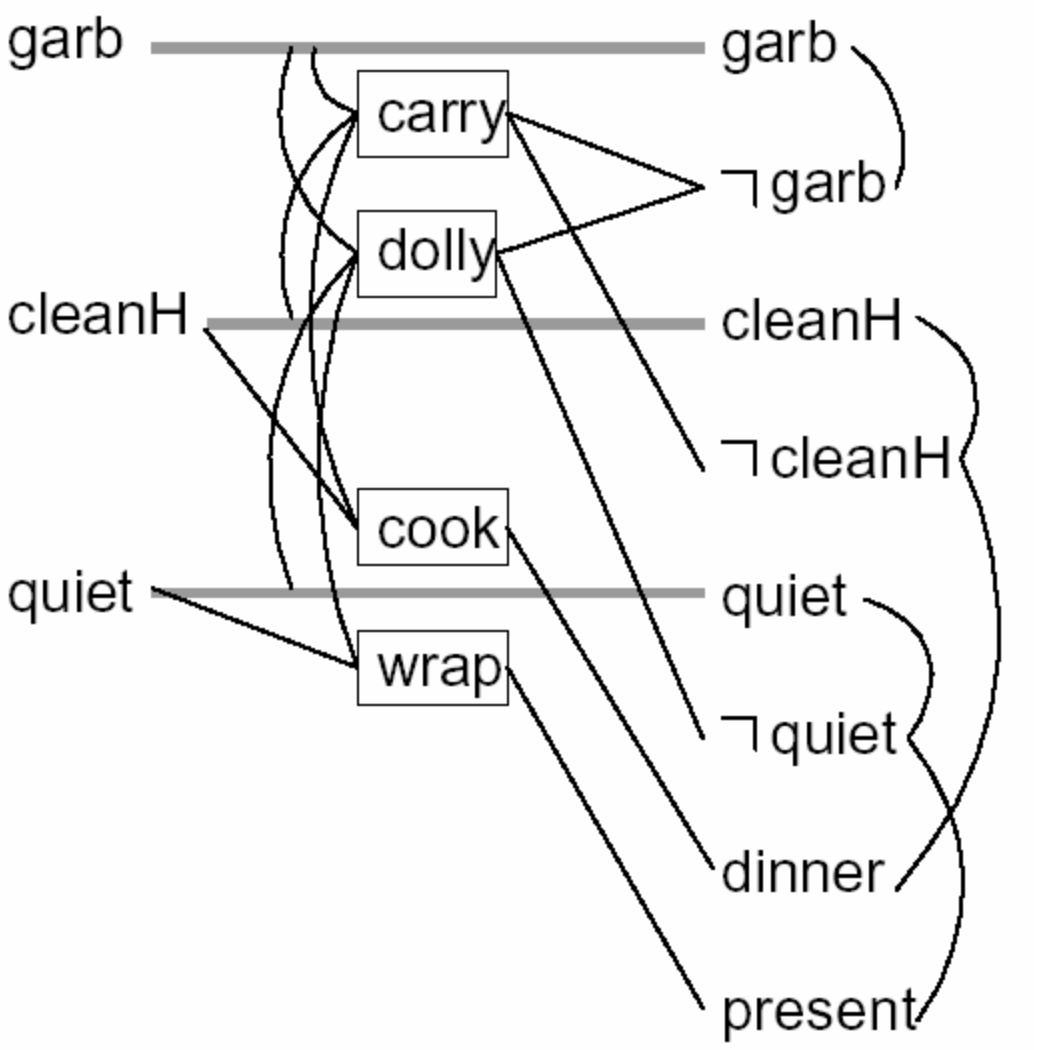
Termination for unsolvable problems

- Graphplan records (memoizes) sets of unsolvable goals:
  - U(i,t) = unsolvable goals at level i after stage t.
- More efficient: early backtracking
- Also provides necessary and sufficient conditions for termination:
  - Assume plan graph levels off at level n, stage t > n
  - If U(n, t-1) = U(n, t) then we know we're in a loop and can terminate safely.

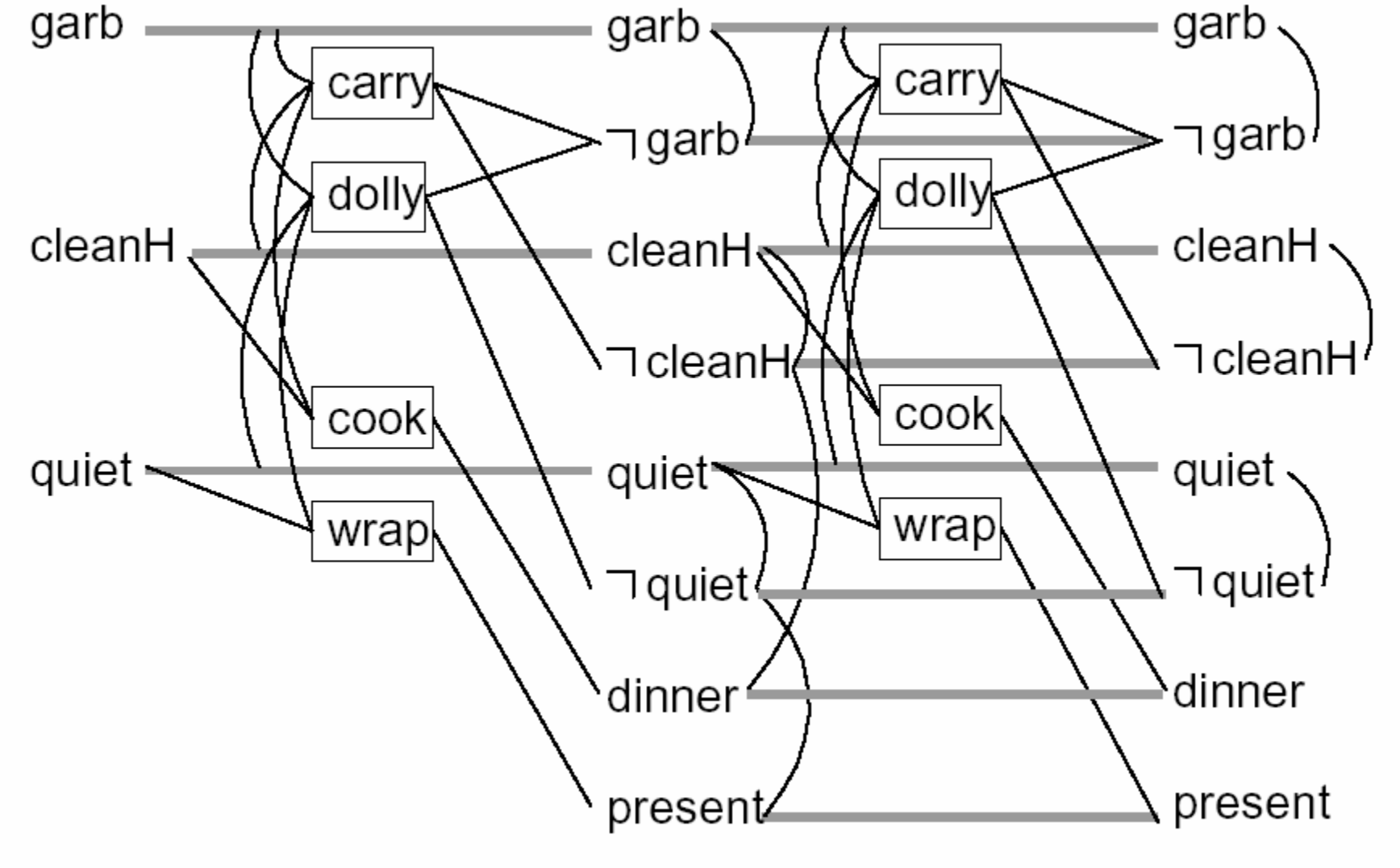
Dinner Date example

- Initial Conditions: (and (garbage) (cleanHands) (quiet))
- Goal: (and (dinner) (present) (not (garbage))
- Actions:
  - Cook :precondition (cleanHands)
    :effect (dinner)
  - Wrap :precondition (quiet)
    :effect (present)
  - Carry :precondition
    :effect (and (not (garbage)) (not (cleanHands))
  - Dolly :precondition
    :effect (and (not (garbage)) (not (quiet)))

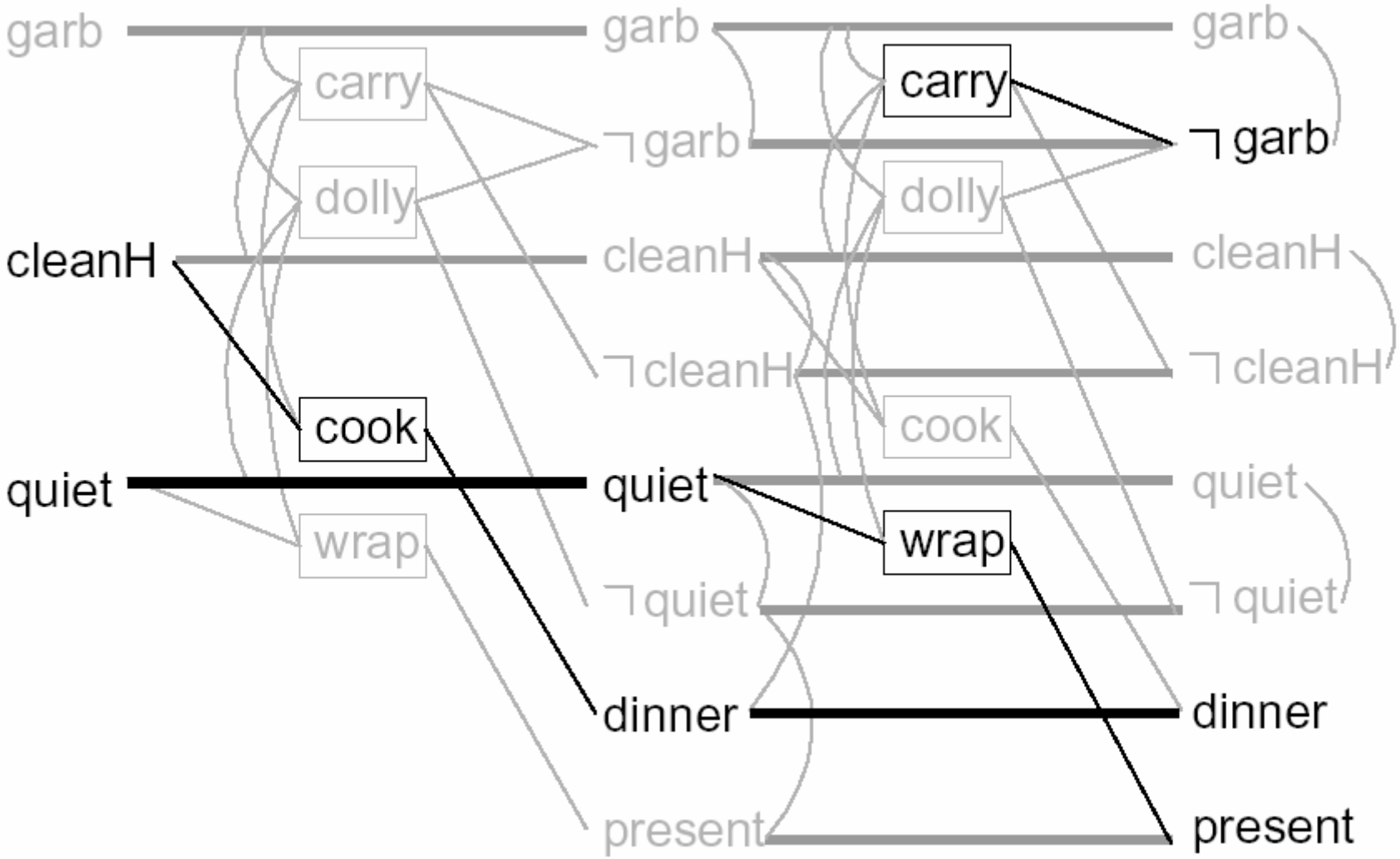
Dinner Date example



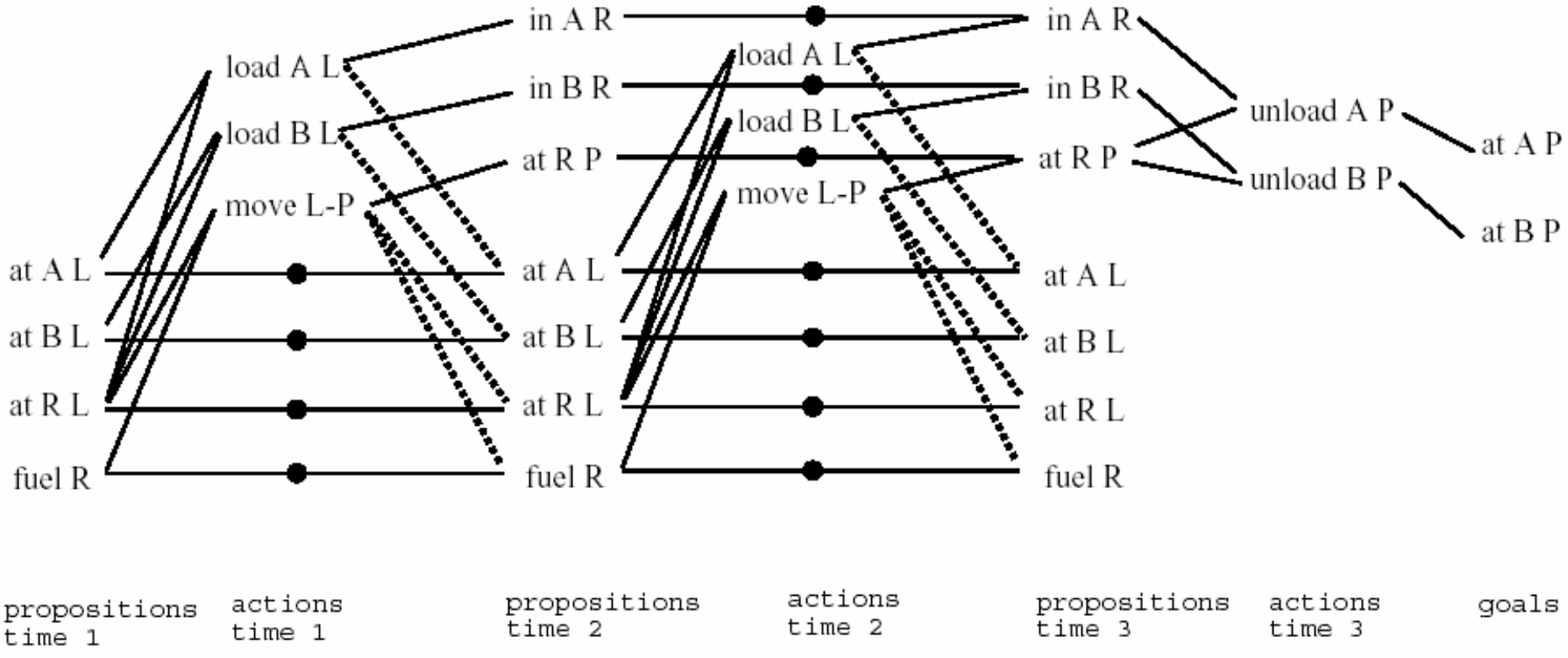
Dinner Date example



Dinner Date example



Planning Graph Example
Rocket problem



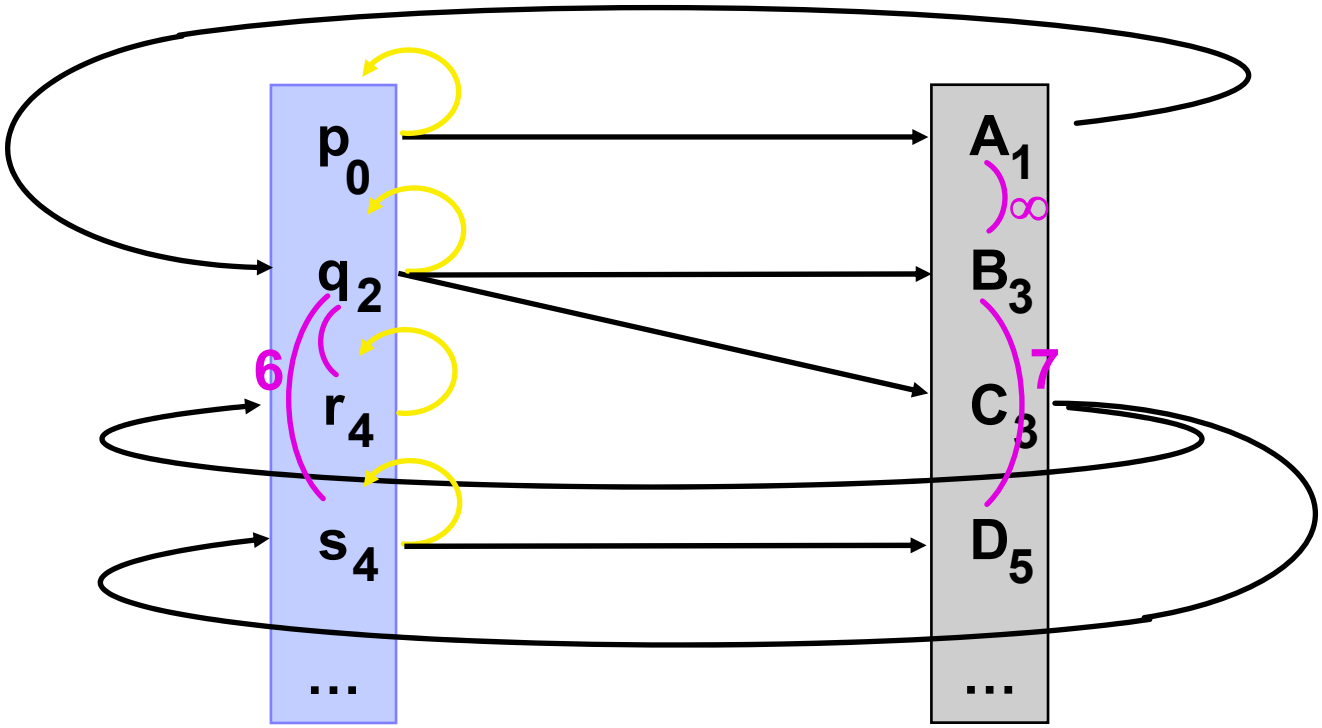
Plan Graph creation is Polynomial

Theorem 1:

- The size of the t-level PG and the time to create it are polynomial in
  - t = number of levels
  - n = number of objects
  - m = number of operators
  - p = propositions in the initial state
- Max nodes proposition level: $O(p+mln^k)$
- Max nodes action level: $O(mn^k)$

k = largest number of action parameters, constant!

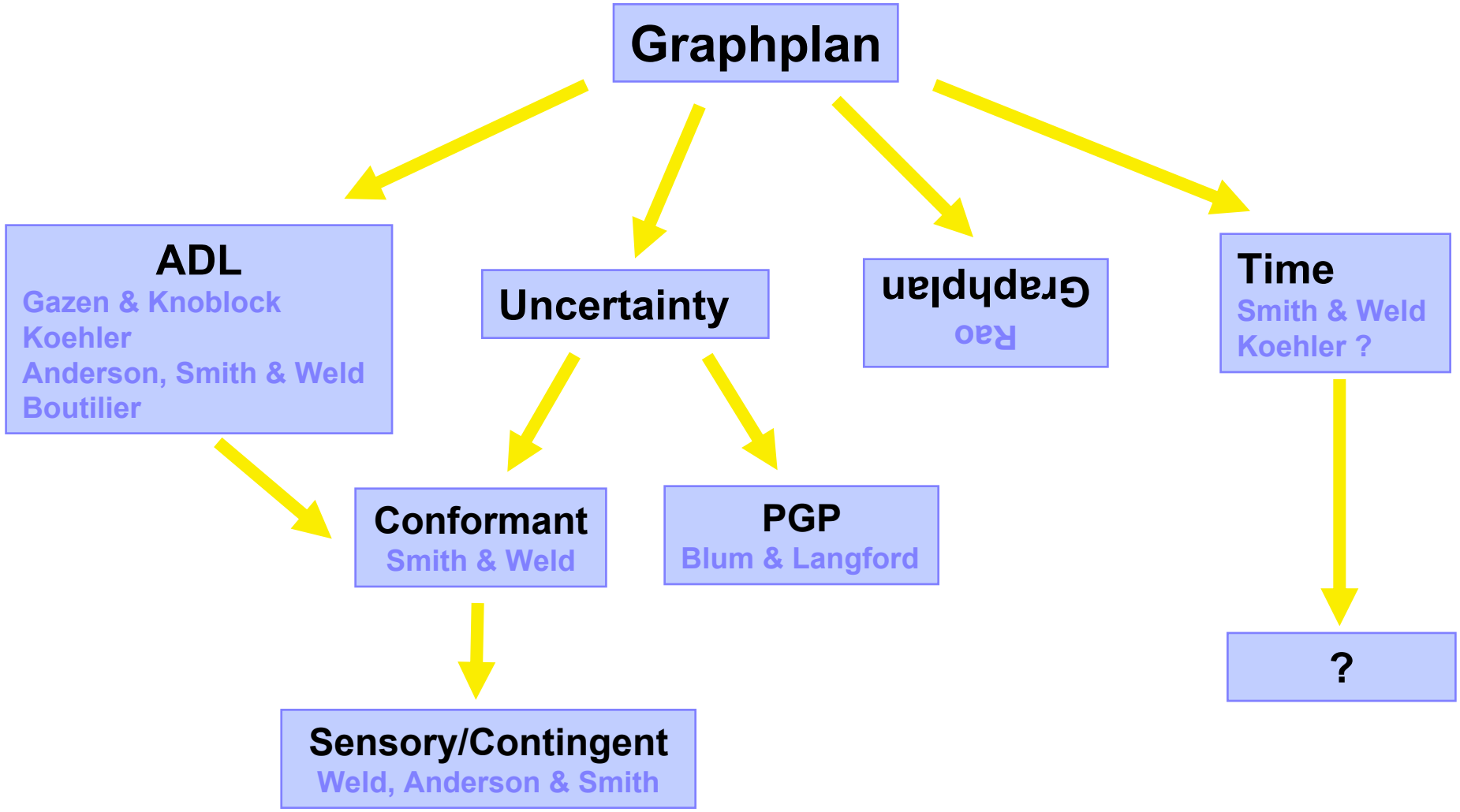
In-place plan graph expansion



Props & actions: start level → start time

Mutex relations: end level → end time

Perverting Graphplan



Expressive Languages

- Negated preconditions
- Disjunctive preconditions
- Universally quantified preconditions, effects
- Conditional effects

Negated Preconditions

- Graph expansion
  - P, ¬P mutex
  - Action deleting P must add ¬P at next level
- Solution extraction

Disjunctive Preconditions

- Convert precondition to DNF
  - Disjunction of conjunctions
- Graph expansion
  - Add action if any disjunct is present, nonmutex
- Solution extraction
  - Consider all disjuncts

Universal Quantification

- Graph Expansion
- Solution Extraction

Universal Quantification

- Graph Expansion
  - Expand action with Herbrand universe
    replace $\forall_{block}\, x \quad P(x)$
    with $P(o_{17}) \wedge P(o_{74}) \wedge \ldots \wedge P(o_{126})$

- Solution Extraction
  - No changes necessary

Conditional Effects

```
move-briefcase (?loc ?new)
    :prec   (and (at briefcase ?loc) (location ?new)
                 (not (= ?loc ?new)))
    :effect (and (at briefcase ?new) (not (at briefcase ?loc))
                 (when (in paycheck briefcase)
                       (and (at paycheck ?new)
                            (not (at paycheck ?loc))))
                 (when (in keys briefcase)
                       (and (at keys ?new)
                            (not (at keys ?loc)))))
```

Full Expansion

```
move-briefcase-empty (?loc ?new)
    :prec   (and (at briefcase ?loc) (location ?new)
                 (not (= ?loc ?new))
                 (not (in paycheck briefcase))
                 (not (in keys briefcase)))
    :effect (and (at briefcase ?new) (not (at briefcase ?loc)))

move-briefcase-paycheck (?loc ?new)
    :prec   (and (at briefcase ?loc) (location ?new)
                 (not (= ?loc ?new))
                 (in paycheck briefcase)
                 (not (in keys briefcase)))
    :effect (and (at briefcase ?new) (not (at briefcase ?loc))
                 (at paycheck ?new) (not (at paycheck ?loc)))

move-briefcase-keys (?loc ?new)
    :prec   (and (at briefcase ?loc) (location ?new)
                 (not (= ?loc ?new))
                 (not (in paycheck briefcase))
                 (in keys briefcase))
    :effect (and (at briefcase ?new) (not (at briefcase ?loc))
                 (at keys ?new) (not (at keys ?loc)))

move-briefcase-both (?loc ?new)
    :prec   (and (at briefcase ?loc) (location ?new)
                 (not (= ?loc ?new))
                 (in paycheck briefcase)
                 (in keys briefcase))
    :effect (and (at briefcase ?new) (not (at briefcase ?loc))
                 (at paycheck ?new) (not (at paycheck ?loc))
                 (at keys ?new) (not (at keys ?loc)))
```

$\overline{\text{in-keys}}$ $\overline{\text{in-pay}}$

in-keys $\overline{\text{in-pay}}$

$\overline{\text{in-keys}}$ in-pay

in-keys in-pay

Factored Expansion

- Treat conditional effects as primitive
  - "component" = <antecendant, consequent> pair
- STRIPS action has one component
- Consider action A
  - Precond: p
  - Effect:

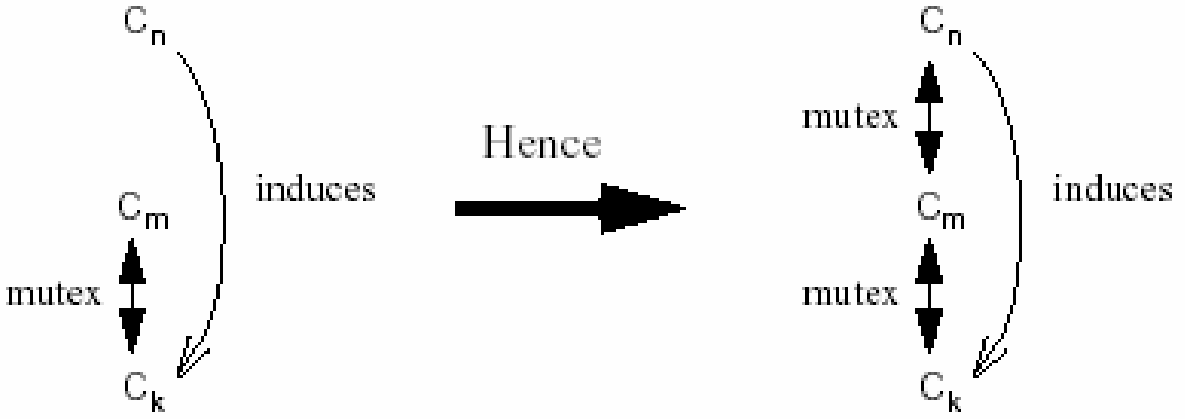
    e
    (when q (f $\wedge \neg$g))
    (when (r $\wedge$ s) $\neg$q)
- A has three components:

antecedent	consequent
p	e
p $\wedge$ q	f $\wedge \neg$g
p $\wedge$ r $\wedge$ s	$\neg$q

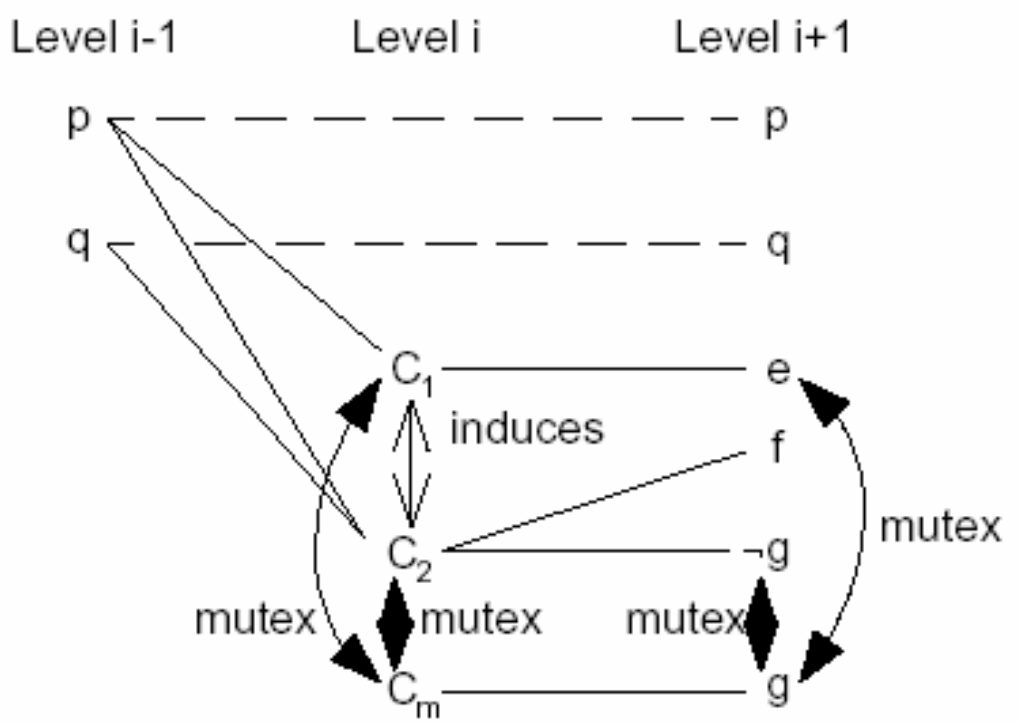
Changes to Expansion

- Components C1 and C2 are mutex at level I if
  - The antecedants of C1 and C2 are mutex at I-1
  - C1, C2 come from *different* action instances, and the consequent of C1 deletes the antecedant of C2, or vice versa
  - ∃ C, C1 *induces* C and C is mutex with C2

- Intuitively, C1 induces C if it is impossible to execute C1 without executing C.
  - C1 and C are parts of same action instance
  - C1 and C aren't mutex (antecedants not inconsistent)
  - The negation of C's antecedant can't be satisfied at level I-1



Induced Mutex



Revised Backchaining

- Confrontation
  - Subgoaling on negation of something

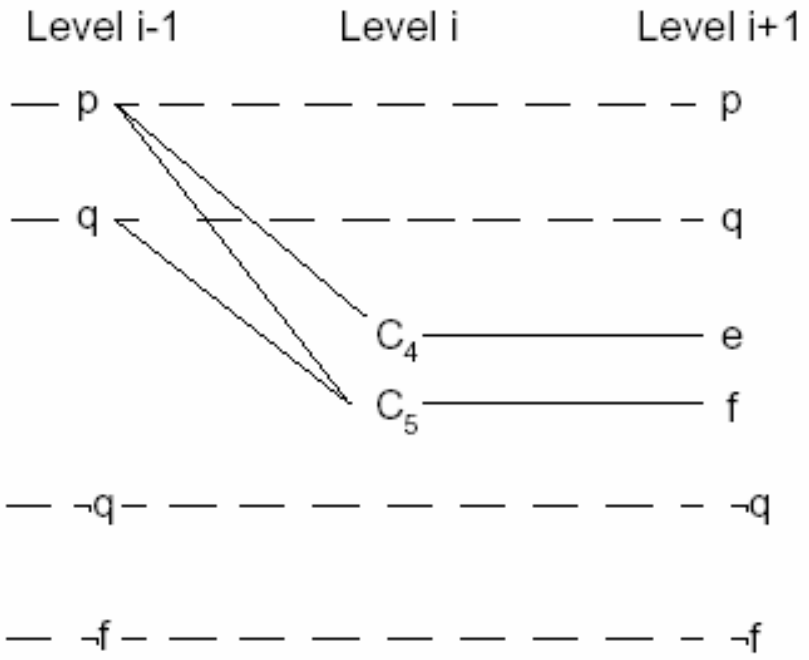


Figure 8: Planning graph when backchaining occurs; in order to prevent C_5 from clobbering $\neg f$ the planner must use confrontation to subgoal on $\neg q$ at level $i-1$.