EECS 496: Sequential Decision Making

Soumya Ray

sray@case.edu

Office: Olin 516

Office hours : T 4-5:30 or by appointment

Recap

- Actions are independent if the ____ effects of one don't interfere with the ____ or _____ effects of another. If two actions are dependent, then the ____ depends on the ____ of _____.
- To track independence, Graphplan uses _____ ______ relations.
- Actions are mutually exclusive if they have (i) inconsistent effects, (ii) interference or (iii) competing needs.
- Propositions are mutually exclusive if they (i) are __________ or (ii) have inconsistent support.
- In the planning graph, (i) propositions ____ by layer, (ii) actions ____ by layer, (iii) mutex relationships ____ by layer.
- Eventually the planning graph ___ ___. This is because _____.
- What is the necessary condition for finding a solution in Graphplan?
- What are the steps of the Graphplan algorithm?
- How do we expand a layer?
- How do we extract a solution?

Today

- Graphplan
- Planning as Satisfiability

Graphplan Algorithm

- Initialize the planning graph with initial state
- While not done
  - *Expand* the planning graph by one level
  - If new level is identical to old level (including mutexes), FAIL
  - If new level satisfies (*), check for solution
  - If found, stop
  - Else go to step 1

Expanding a Planning Graph

- Add an action layer
  - If all preconditions are in the previous layer and all nonmutex, add the action
- Add proposition layer
  - Add all effects of all actions in the action layer (including maintenance actions)
- Add action mutexes for new layer
- Add proposition mutexes for new layer

Solution Extraction

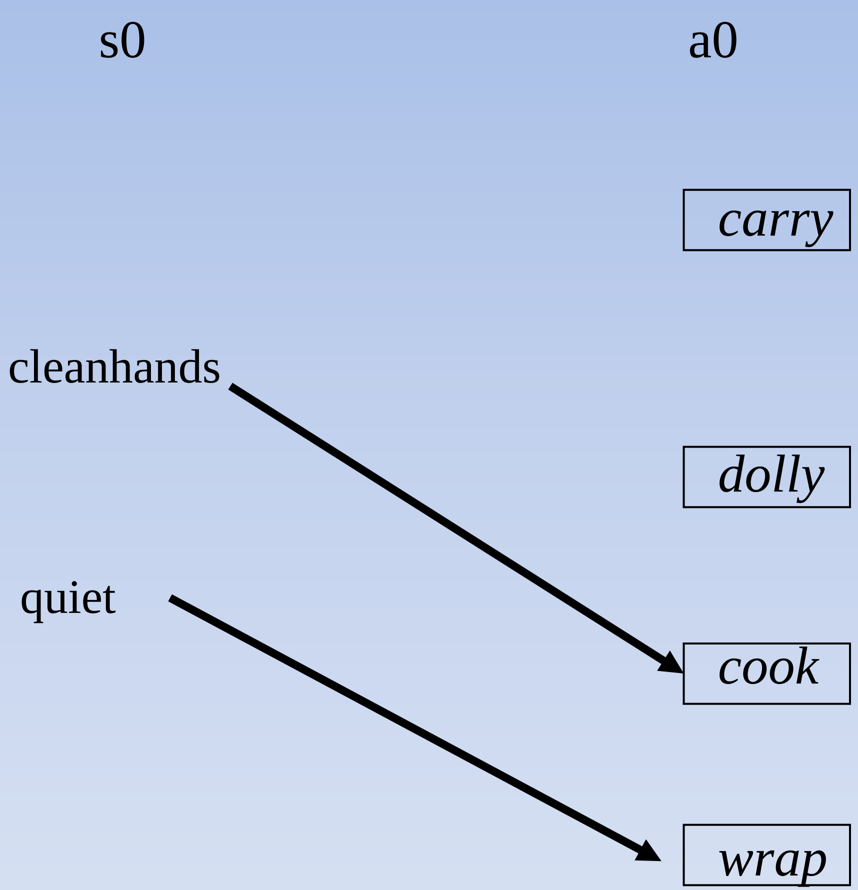
- If goals present and non-mutex
  - Choose any actions that satisfy the goals
  - Add actions' preconditions to new goals
  - Repeat until initial state is reached, or some level is reached where mutex relations hold

Example – Dinner Date

- Initial State: {cleanHands, quiet}
- Goal: {dinner, present, noGarbage}

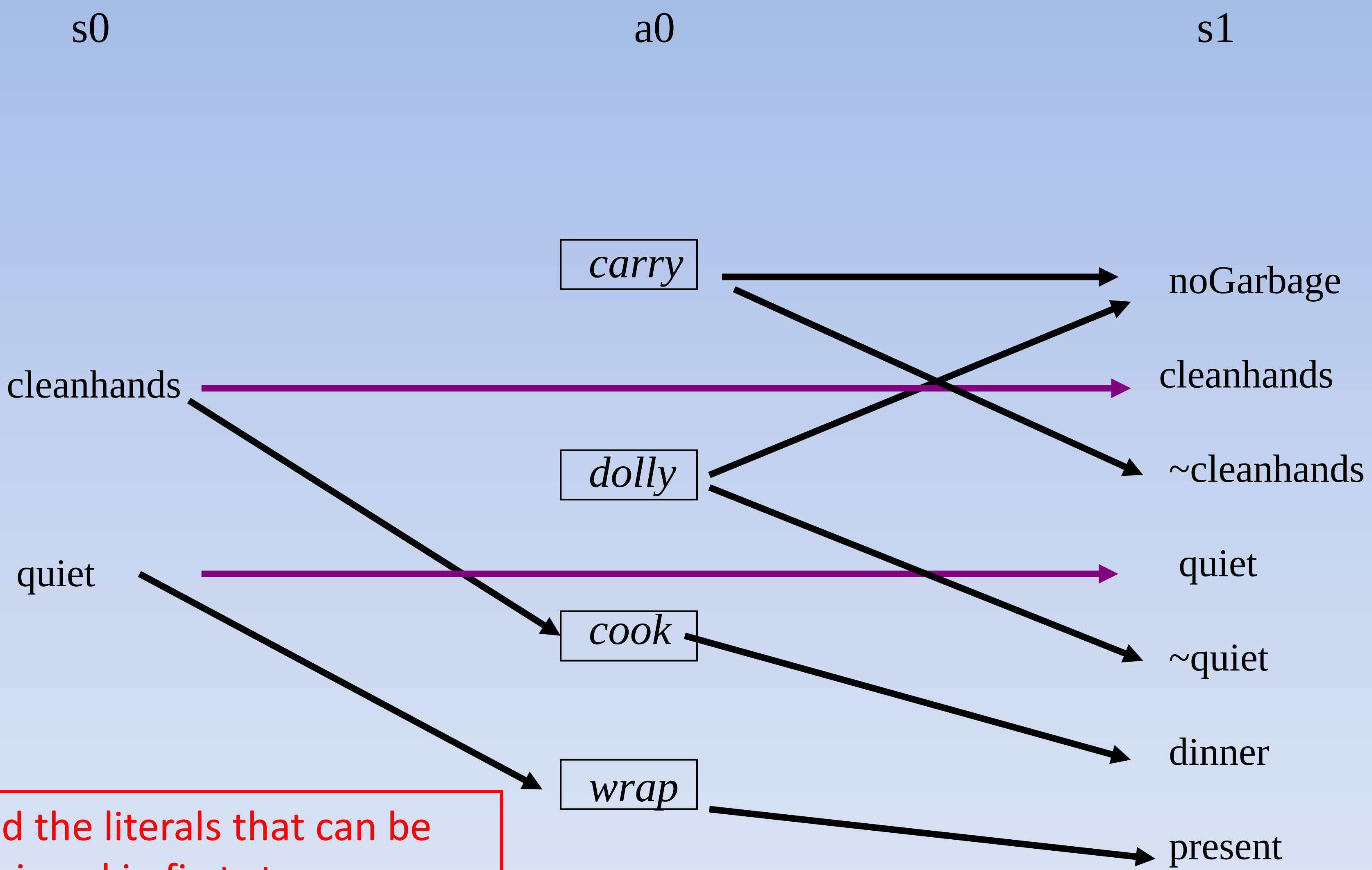
Action	Preconditions	Effects
cook	cleanHands	dinner
wrap	quiet	present
carry	*none*	noGarbage, ¬cleanHands
dolly	*none*	noGarbage, ¬quiet

Example – Plan Graph Construction



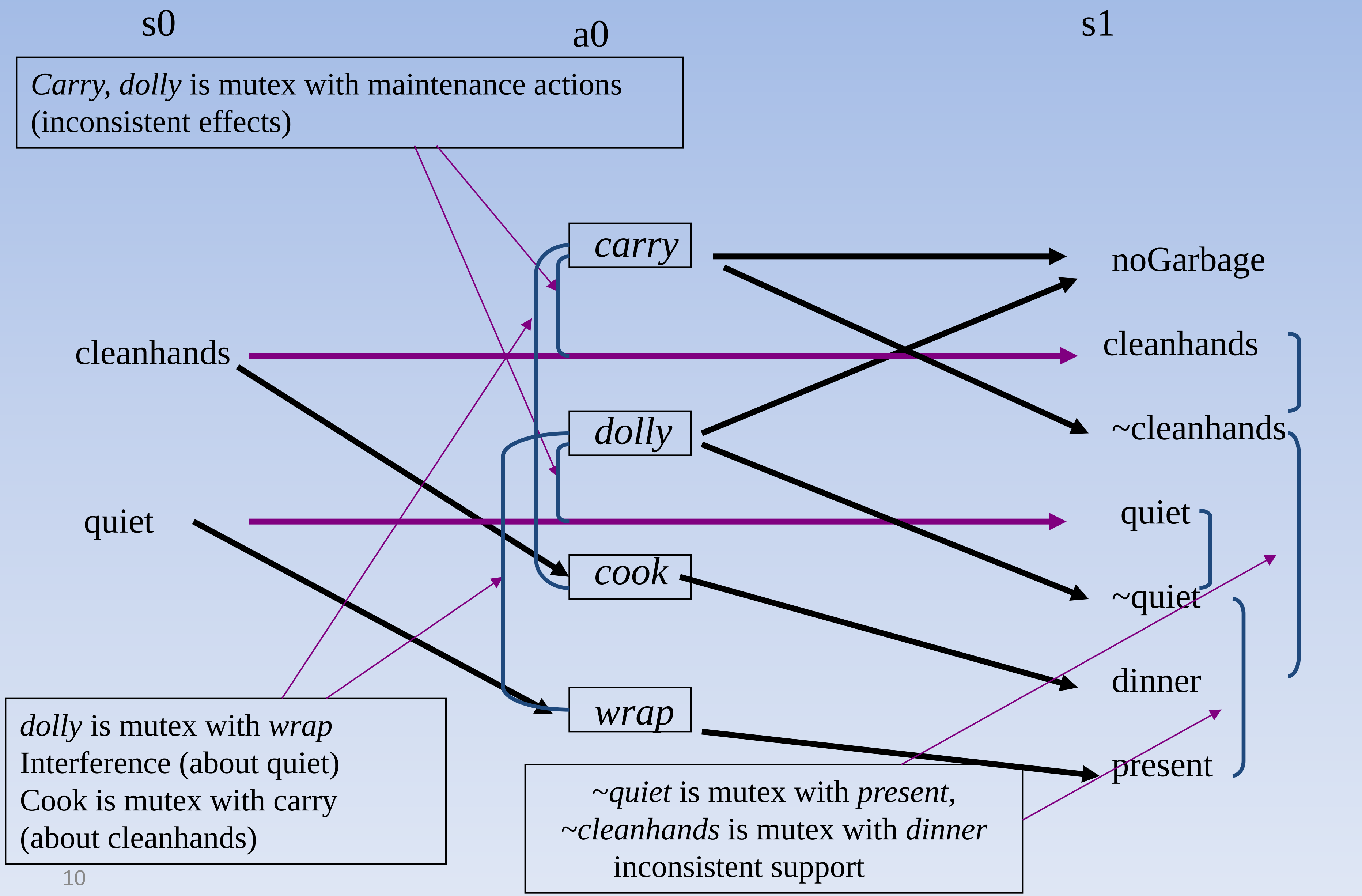
Add the actions that can be executed in initial state

Example - continued



Add the literals that can be achieved in first step

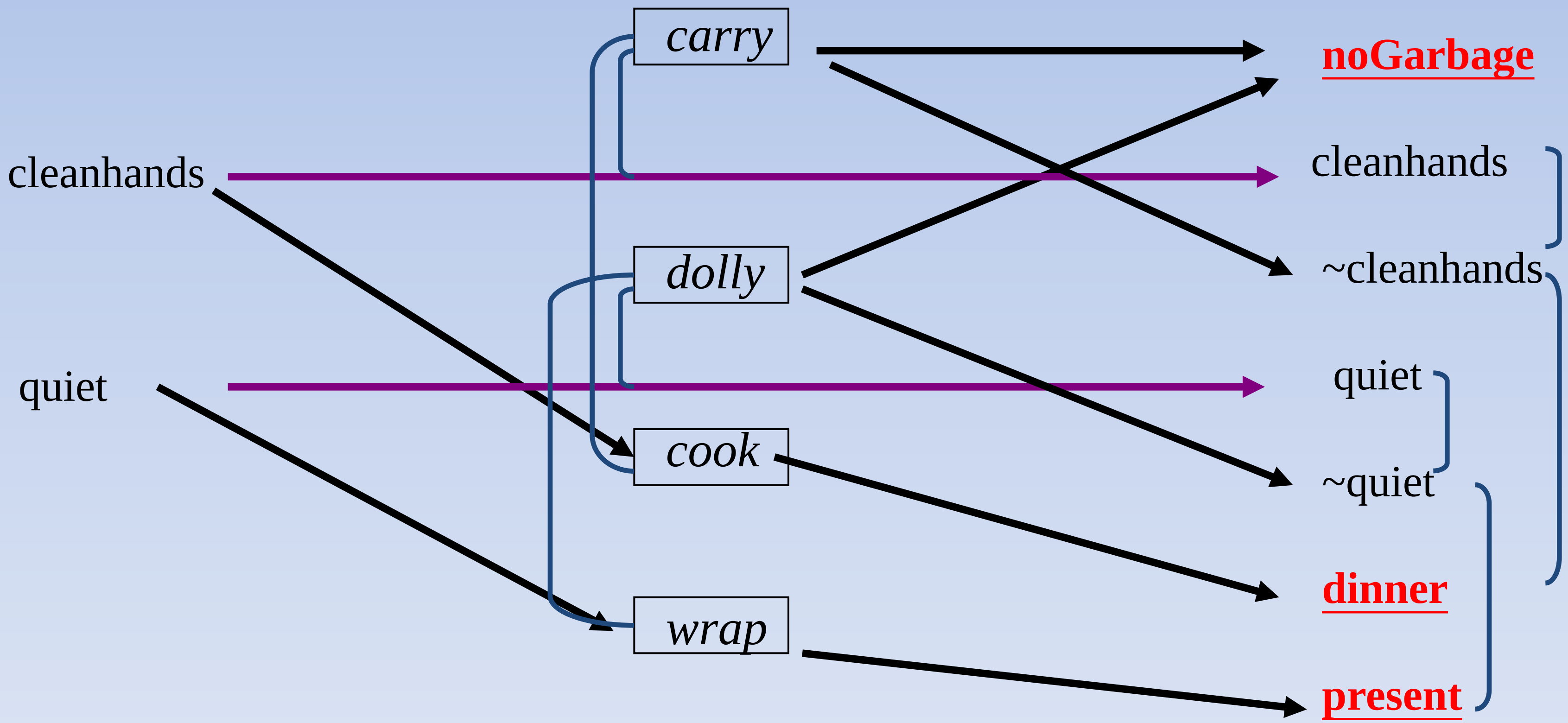
Example - continued



Do we have a solution?

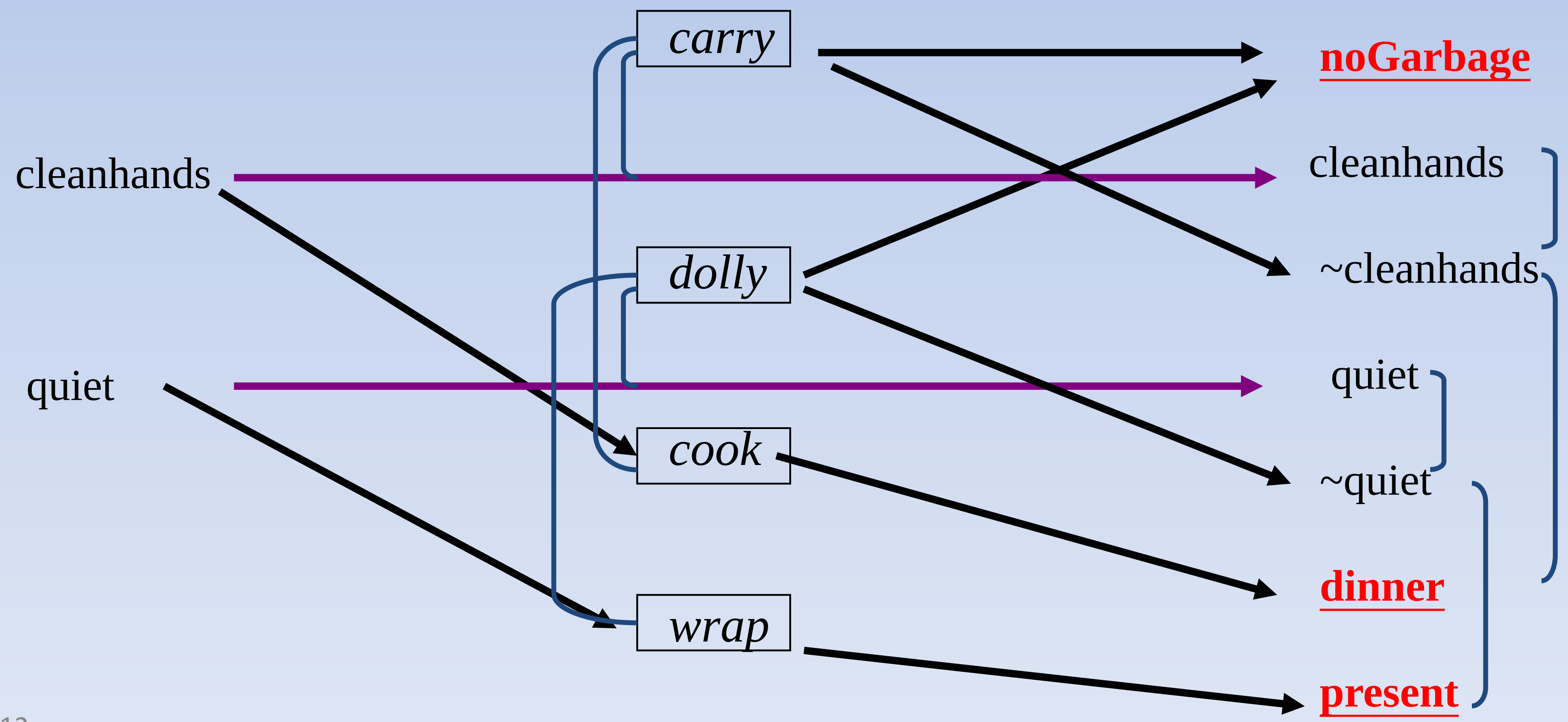
The goal is: **{noGarbage, dinner, present}**

All are possible in layer s1
None are mutex with each other
There is a chance that a plan exists
Now try to find it – solution extraction

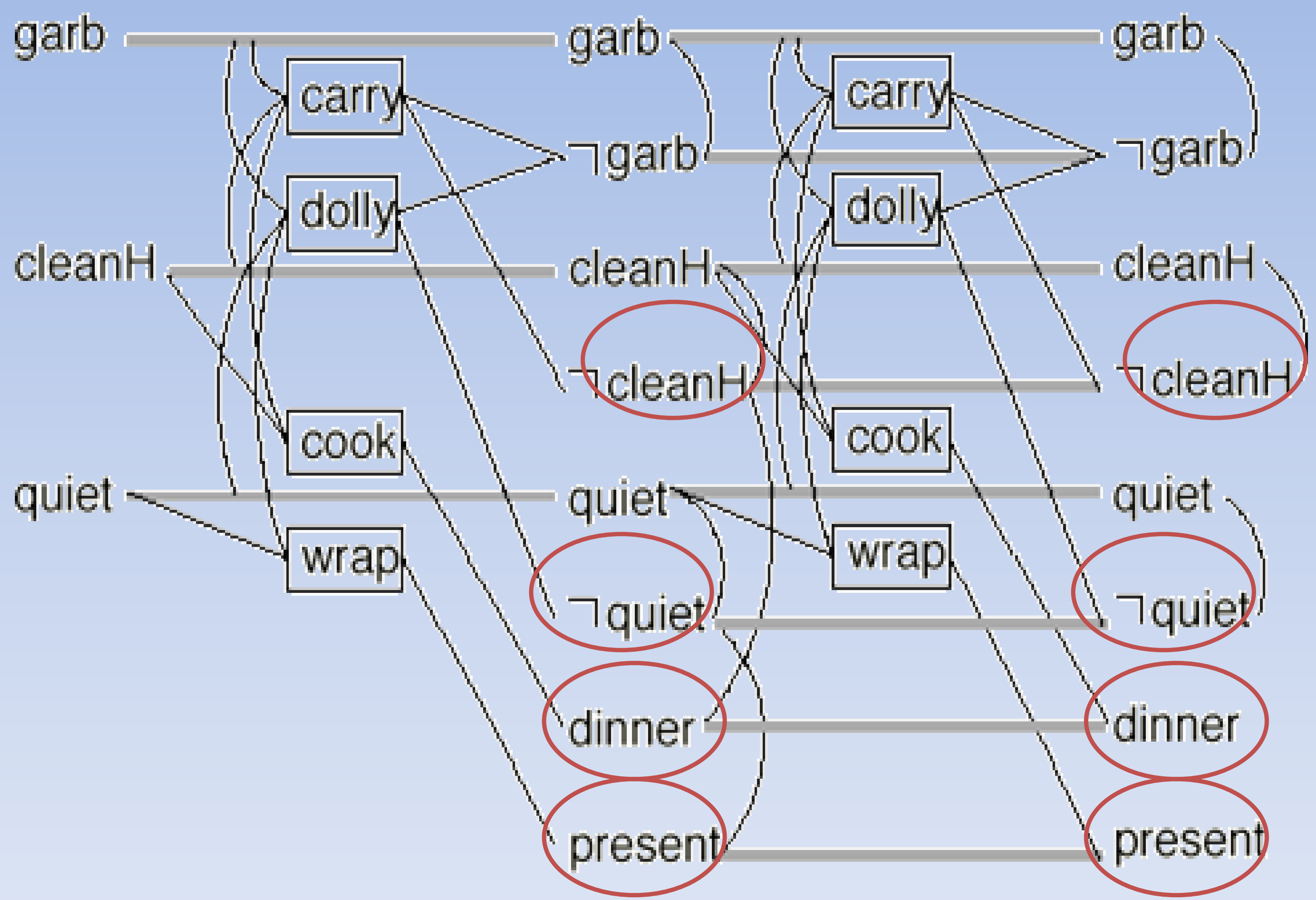


Possible solutions

- Two possible sets of actions for the goals at layer s1:
  {wrap, cook, dolly} and {wrap, cook, carry}
- Neither set works -- both sets contain actions that are mutex

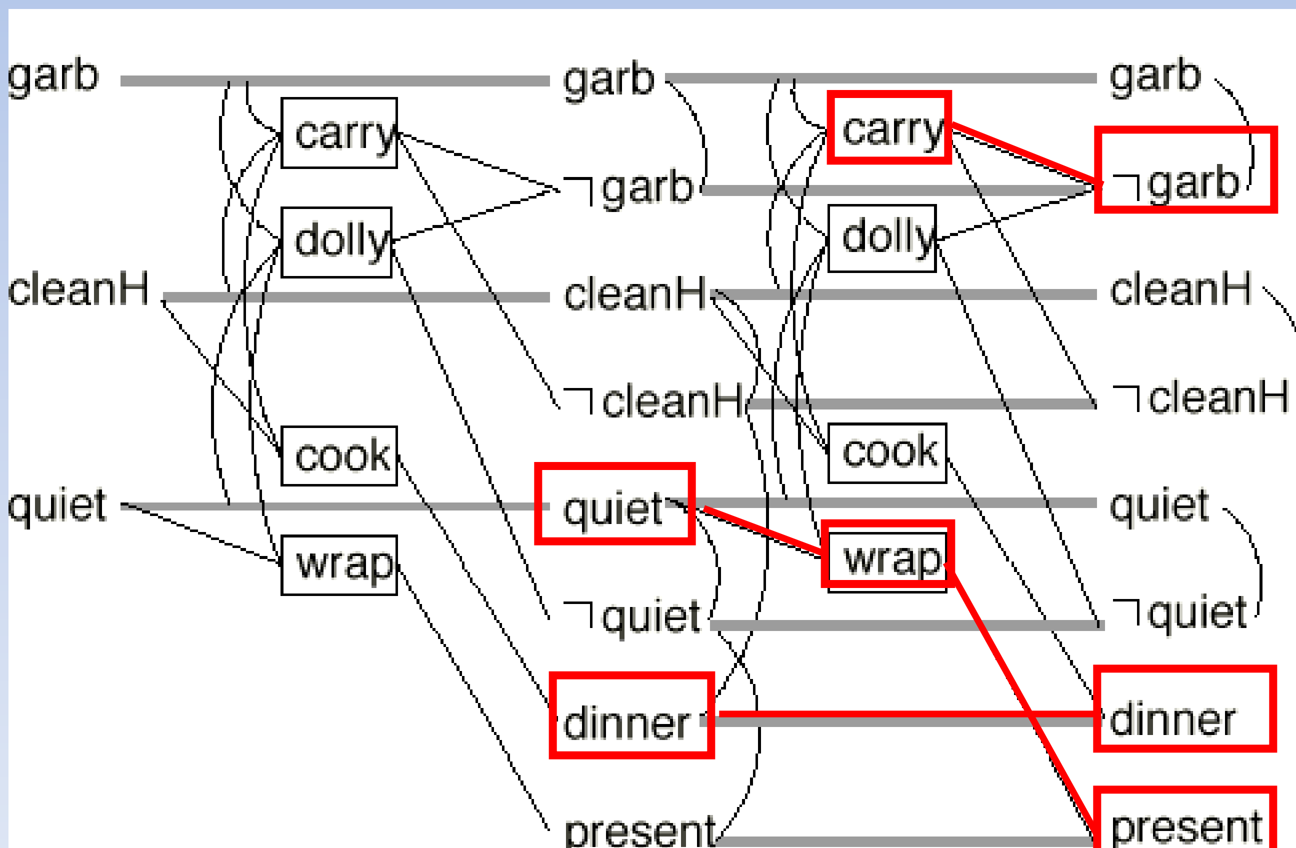


Add new layer… *Note: noGarbage is shown as ¬garb*



Do we have a solution?

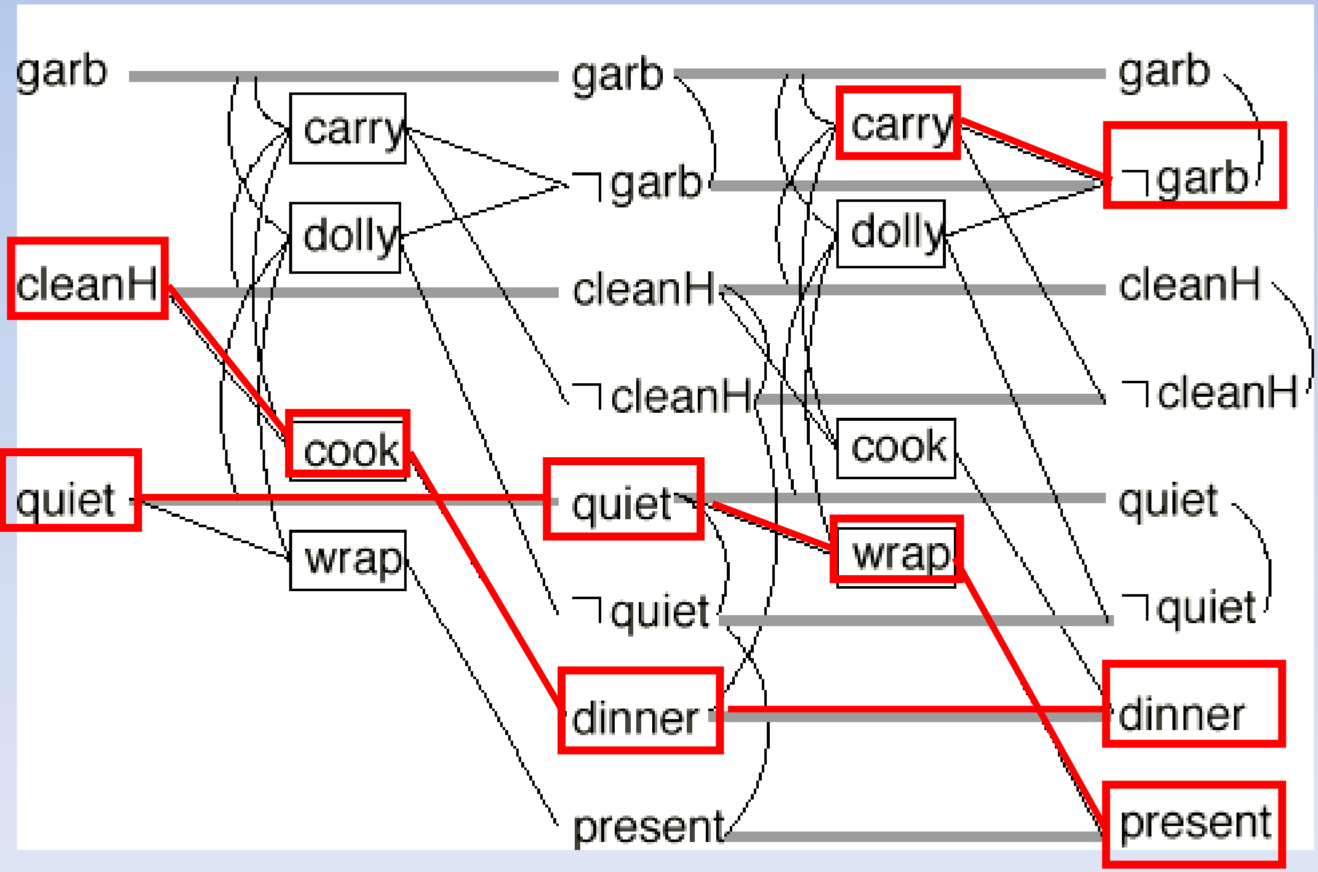
Several action sets look OK at layer 2
Here's one of them
We now need to satisfy their preconditions



Do we have a solution?

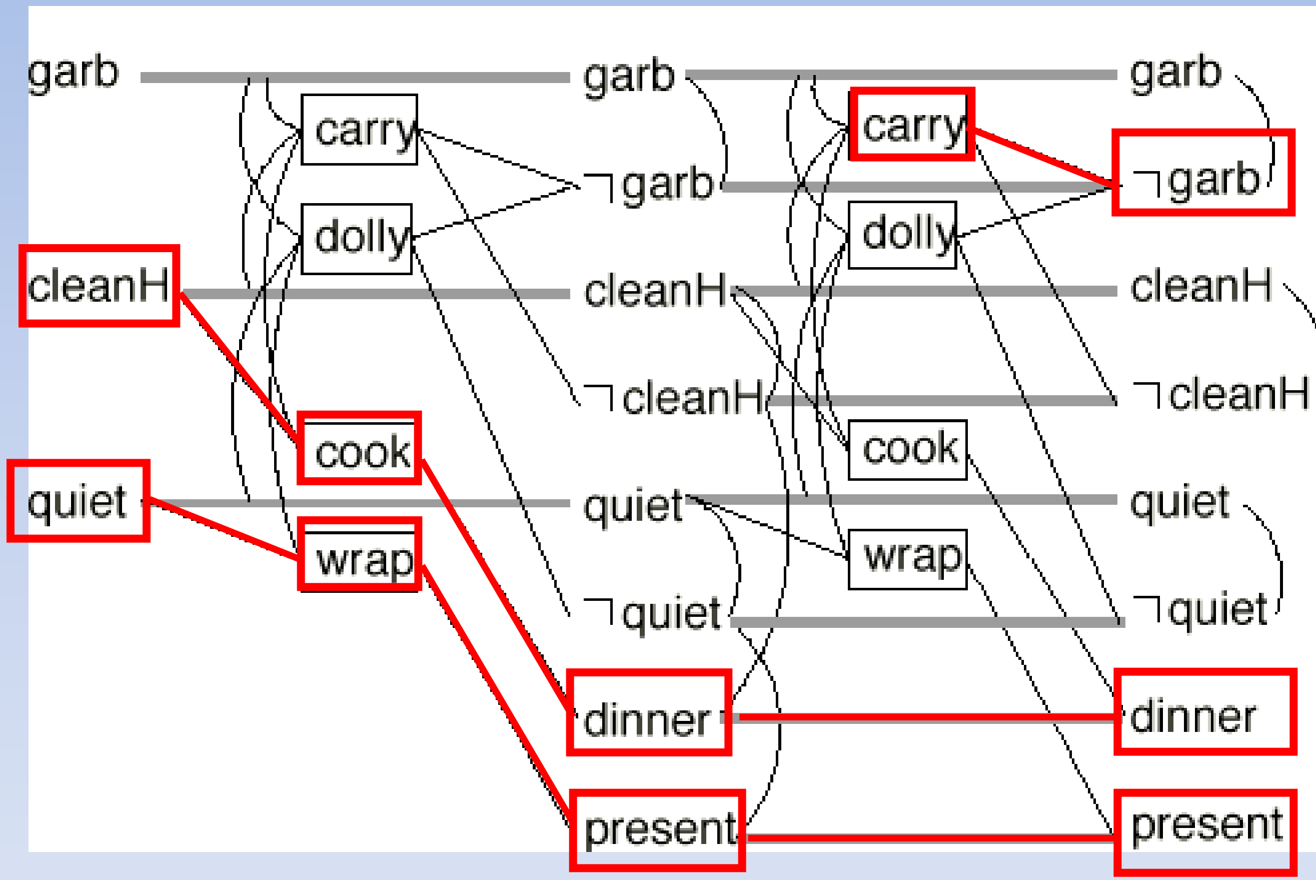
The action set {cook, quiet} at layer 1 supports preconditions
Their preconditions are satisfied in initial state
So we have found a solution:

{cook} ; {carry, wrap}



Another solution:

{cook,wrap} ; {carry}



Planning Graphs in Forward Search

- A planning graph is a general data structure that tracks different constraints of the problem
  - Necessarily unreachable literals and actions
  - Possibly reachable literals and actions
  - Mutually exclusive literals and actions

- Once we build this structure, we can use it in many ways
  - E.g. to derive heuristics for forward search

The FF Heuristic

- Recall that we could use relaxed plans to compute an admissible heuristic for forward planning
- The FF ("Fast Forward") planner constructs a planning graph with the relaxed actions and uses the length of the found plan as a heuristic
- Further, the actions selected by Graphplan in the resulting plan can be prioritized for exploration

Planning as Satisfiability

- The STRIPS representation is a fragment of first order logic and planners are essentially logical inference algorithms

- We can use this connection in a stronger way by framing planning as a satisfiability problem

Situation Calculus

- This is not new, in fact, the first approaches to solving planning problems used this idea
  - But in full FOL
  - Then they realized they had a better chance of winning the lottery than this procedure had of being practical and gave up
- Restricting the language to being propositional, coupled with the development of fast SAT solvers, renewed development along these lines

Planning as Satisfiability

- Suppose we "compile" a planning problem into a propositional formula
- Then we can employ very fast propositional SAT solvers for planning
  - This is the basis of the SATPlan algorithm

Recall: Literals, Clauses and CNF

- A ***literal*** is either a proposition or the negation of a proposition
- A ***clause*** is a disjunction of literals
- A formula is in ***conjunctive normal form (CNF)*** if it is the conjunction of clauses
  - $(\neg R \vee P \vee Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee R)$
- Any formula can be represented in conjunctive normal form (CNF)

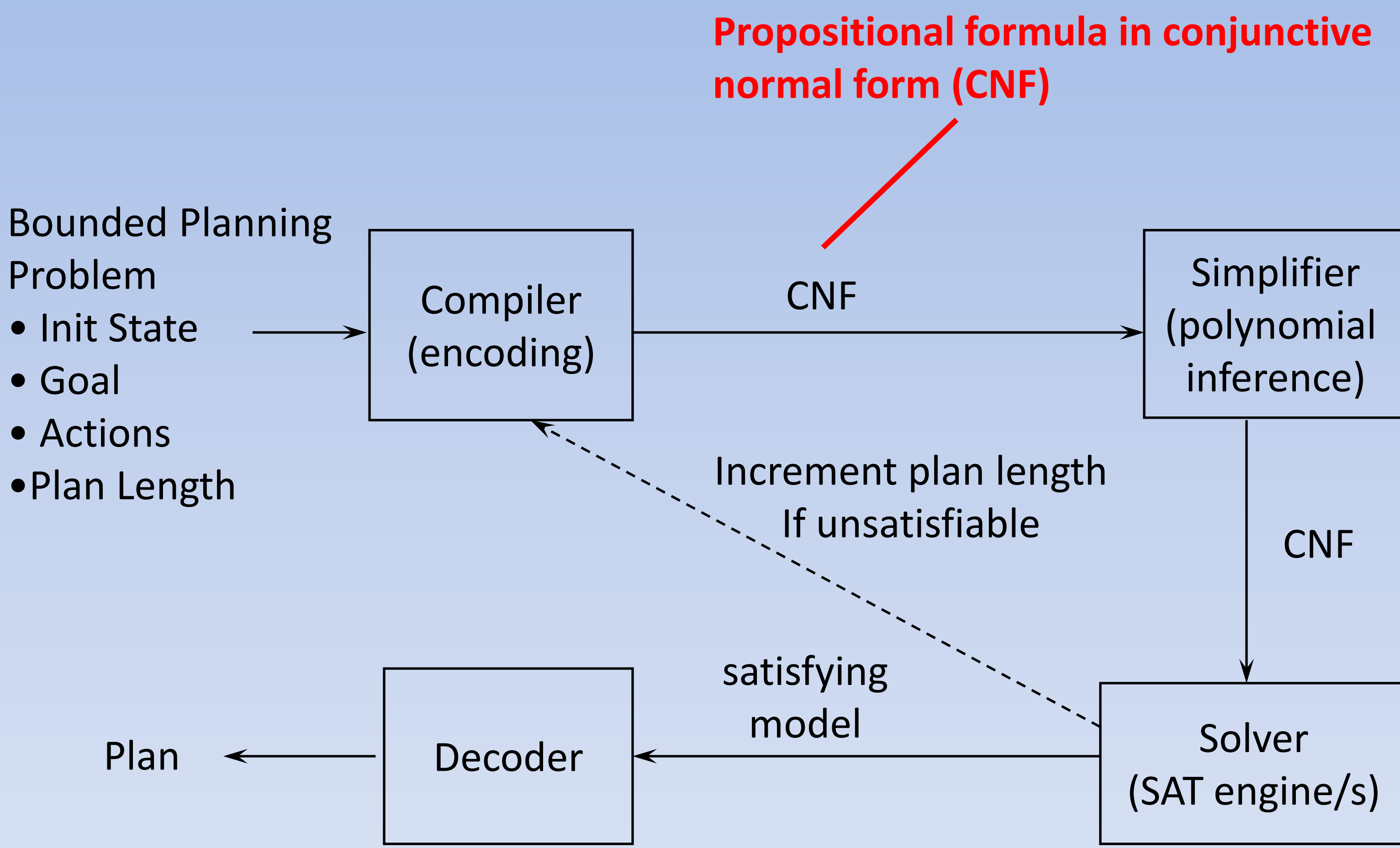
WalkSAT (Kautz and Selman, 1993)

- Start with initial complete assignment
- Repeat MAX_FLIPS times:
  - If all clauses true, return current assignment
  - Else, pick random unsatisfied clause
  - With small probability ε, flip the assignment of a random variable in clause
  - Else flip the assignment of the variable in clause that maximizes satisfied clauses

Planning as Satisfiability: Key idea

- ***Bounded*** planning problem (*P,n*):
  - *P* is a planning problem; *n* is a positive integer
  - Find a solution for *P* of length *n*
- Create a propositional CNF formula that represents:
  - Initial state
  - Asserts goal after *n* time steps
  - Action dynamics for *n* time steps
- Such that:
  1) ***any*** satisfying truth assignment of the formula represents a solution to (*P*,*n*)
  2) if (*P*,*n*) has a solution then the formula is satisfiable

Architecture of a SAT-based Planner



Optimality and Failure Termination

- With a complete satisfiability tester, this approach will produce optimal plans for solvable problems

- We can use a Graphplan analysis to determine an upper bound on *n*, giving a way to detect unsolvability

Planning → SAT

- How to describe states?

- How to describe actions?

State Representation

- States in STRIPS are conjunctions of unnegated, ground, function-free literals
  - All conditions that hold in that state
  - *Block(A), Block(B), On(A,B), On(B, Table), GripperEmpty*
  - The “Closed World Assumption” is used

State Representation

- We can use the same literals, but we also need to indicate *the time-indexed state at which this literal should hold*
  - So instead of $On(A,B)$ we have $On(A,B,s0)$
  - Each such literal is treated as a single proposition by the inference engine ($On\_A\_B\_s0$)
  - These are called *fluents*, because they vary with time (states)

State Representation

- A general purpose inference procedure does not have CWA, so we also need to specify all fluents that are *false* at a state

- Final state representation for state S_i:

$$\left(\bigwedge_{f_j \in S_i} f_{ji} \right) \wedge \left(\bigwedge_{f_j \notin S_i} \neg f_{ji} \right)$$

Representing the Goal

- We know that the n^{th} state must satisfy the goal
- Also the goal is just a set of positive literals, so we represent the goal as:

$$\left(\bigwedge_{g \in Goal} g_n \right)$$

Action Representation

- We write a formula that describes what needs to have happened if the i^{th} action in the plan is a_i

$$a_i \Rightarrow \left[\left(\bigwedge_{p_j \in \text{precond}(a)} p_{ji} \right) \wedge \left(\bigwedge_{e_j \in \text{effects}(a)} e_{j,i+1} \right) \right]$$

- Need one such formula for every action *for every step*
- Do we need anything else?