Planning and Search

Classical Planning: Planning graphs, Graphplan

Outline

- ♢ Propositionalising planning problems
- ♢ Planning graphs
- ♢ Heuristics
- ♢ GraphPlan algorithm

Propositional planning problems

Both GraphPlan and SATPlan (in the next lecture) apply to propositional planning problem

How to propositionalise PDDL action schema: replace each action schema with a set of all its ground (no variables) instances.

For example, replace

$Move(b, x, y)$:
PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)$

with

$Move(A, B, C)$:
PRECOND: $On(A, B) \wedge Clear(A) \wedge Clear(C)$
EFFECT: $On(A, C) \wedge Clear(B) \wedge \neg On(A, B) \wedge \neg Clear(C)$

$Move(A, C, B)$:

PRECOND: $On(A, C) \wedge Clear(A) \wedge Clear(B)$

EFFECT: $On(A, B) \wedge Clear(C) \wedge \neg On(A, C) \wedge \neg Clear(B)$

$Move(B, A, C)$:

PRECOND: $On(B, A) \wedge Clear(B) \wedge Clear(C)$

EFFECT: $On(B, C) \wedge Clear(A) \wedge \neg On(B, A) \wedge \neg Clear(C)$

and so on.

Ground fluents can be treated as propositional symbols: $On(A, B)$ as something like p_{AB}.

Planning graphs

GraphPlan is an algorithm based on planning graph

Planning graphs are also used as a source of heuristics (an estimate of how many steps it takes to reach the goal)

Planning graph is an approximation of a complete tree of all possible actions and their results

Planning graph 2

Planning graph is organised into **levels**

Level S_0: initial state, consisting of nodes representing each fluent that holds in S_0

Level A_0: each ground action that might be applicable in S_0

Then alternate S_i and A_i

S_i contains fluents which $could$ hold at time i, (may be both P and $\neg P$); literals may show up too early but never too late

A_i contains actions which $could$ have their preconditions satisfied at i

Example

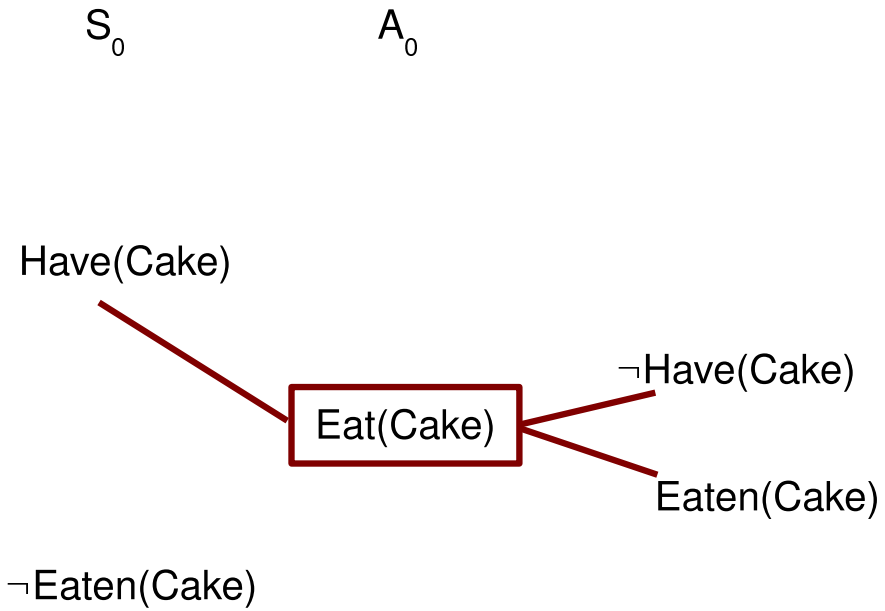
Initial state: $Have(Cake)$

Goal: $Have(Cake) \wedge Eaten(Cake)$

$Eat(Cake)$:
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake)$

$Bake(Cake)$:
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$

(Incomplete) planning graph example



Building a planning graphs

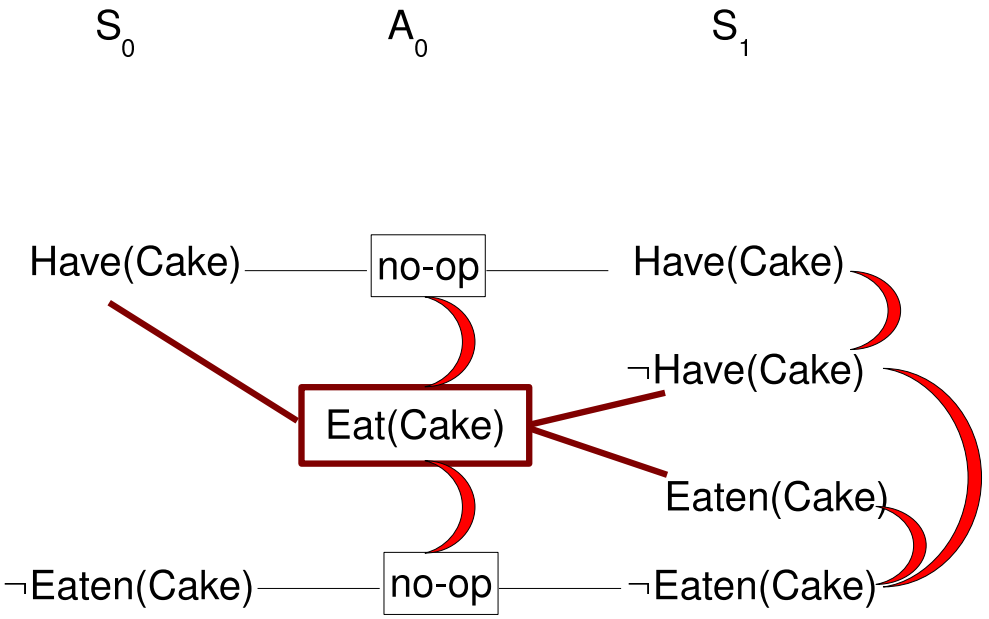
In addition to 'normal' action, persistence action or no-op (no operation): one for each fluent, preserves the fluent's truth.

Mutex or mutual exclusion links depicted by red semicircles mean that actions cannot occur together

Similarly there are mutex links between fluents.

Build the graph until two consecutive levels are identical; then the graph levels off.

Another incomplete planning graph example



Mutex between actions

Mutex relation holds between two actions at the same level if any of the following three conditions holds:

♢ Inconsistent effects: one action negates an effect of another. For example, $Eat(Cake)$ and persistence for $Have(Cake)$ have inconsistent effects ($\neg Have(Cake)$ and $Have(Cake)$)

♢ Interference: one of the effects of one action is the negation of a precondition of the other. For example $Eat(Cake)$ interferes with the persistence of $Have(Cake)$ by negating its precondition

♢ Competing needs: one of the preconditions of one action is mutually exclusive with a precondition of the other. For example, $Eat(Cake)$ has precondition $Have(Cake)$ and $Bake(Cake)$ has precondition of $\neg Have(Cake)$.

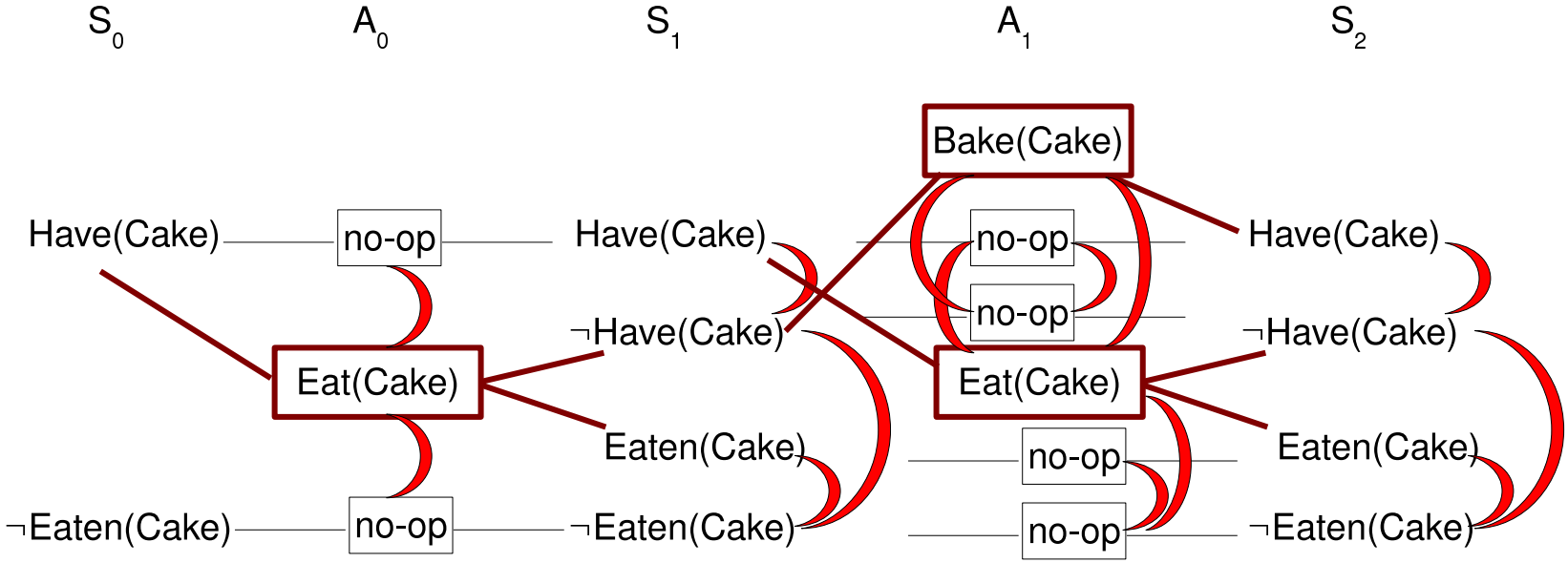
Mutex between fluents

Mutex holds between fluents if:

$\diamondsuit$ they are negations of each other, like $Have(Cake)$ and $\neg Have(Cake)$

$\diamondsuit$ each possible pair of actions that could achieve the two literals is mutually exclusive, for example $Have(Cake)$ and $Eaten(Cake)$ in S_1 can only be achieved by persistence for $Have(Cake)$ and by $Eat(Cake)$ respectively. (In S_2 can use persistence for $Eaten(Cake)$ and $Bake(Cake)$ which are not mutex).

Planning graph example



Size of the planning graph

Polynomial in the size of the problem (unlike a complete search tree, which is exponential!)

If we have n literals and a actions,

each S_i has no more than n nodes and n^2 mutex links,

each A_i has no more than $a + n$ nodes (n because of no-ops), $(a + n)^2$ mutex links, and $2(an + n)$ precondition and effect links.

Hence, a graph with k levels has $O(k(a + n)^2)$ size.

Using planning graph for heuristic estimation

If some goal literal does not appear in the final level of the graph, the goal is not achievable

The cost of achieving any goal literal can be estimated by counting the number of levels before it appears

This heuristic never overestimates

It underestimates because planning graph allows application of actions (including incompatible actions) in parallel

Conjunctive goals:

max level heuristic: max level for any goal conjunct (admissible but inaccurate)

set level heuristic: which level they all occur on without mutex links (better, also admissible)

FastForward

Example of a planning system using planning graphs for heuristics

FF or FastForward system (Hoffman 2005)

Forward space searcher

Ignore-delete-lists heuristic (see lecture 9) estimated using planning graph

Uses hill-climbing search with this heuristic to find solution

When hits a plateau or local maximum uses iterative deepening to find a better state or gives up and restarts hill-climbing

GraphPlan

GraphPlan repeatedly adds a level to a planning graph with EXPAND-GRAPH.

Once all the goals show up as non-mutex in the graph, calls EXTRACT-SOLUTION on the graph to search for a plan.

If that fails, extracts another level.

GraphPlan algorithm

```
function GraphPlan(problem) returns solution or failure
    graph ← Initial-Planning-Graph(problem)
    goals ← Conjuncts(problem.Goal)
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← Extract-Solution(graph,goals, NumLevels(graph)
            if solution ≠ failure then return solution
            else if No-solution-possible(graph) then return failure
        graph ← Expand-Graph(graph,problem)
```

Extracting solution

Backward search problem:

The initial state is the last level of the planning graph, S_n, along with the set of goals

Available actions in S_i: any set of conflict-free actions in A_{i-1} whose effects cover the goals in the state. Conflict-free means: no two actions are mutex and no two of their preconditions are mutex.

The result of applying it is a subset of S_{i-1} which has as its set of goals the preconditions of the selected set of actions.

The goal is to reach a state at level S_0 such that all the goals are satisfied.

The cost of each action is 1.

Heuristic

The search may still degenerate to an exponential exploration

Heuristic:

1. Pick the literal with a highest level cost

2. To achieve this literal, pick actions with easiest preconditions (the set of preconditions which has the smallest max level cost)

Termination of GraphPlan

The graph will level off (assuming a finite set of literals and actions):

♢ Literals in planning graphs increase monotonically (because of persistence)

♢ Actions increase monotonically (because preconditions don't go away)

♢ Mutexes decrease monotonically (if two actions are mutex at level t, they were mutex at all previous levels)

When the graph levels off, if it is missing one of the goal conjuncts or two of the goals are mutex, solution is impossible and GraphPlan returns failure.

If the graph levels off but EXTRACT-SOLUTION fails to find a solution, we mey need to expand the graph finitely many times (but this will terminate - uses additional tricks).

Next lecture

SATPLan

Complexity of classical planning

Successful classical planning systems

Russell and Norvig 3rd edition chapter 10

Russell and Norvig 2nd edition chapter 11