Plan graphs & GraphPlan & SATPlan

Chapter 11.4-11.7

Some material adapted from slides by Jean-Claude Latombe / Lise Getoor

GraphPlan: Basic idea

- Construct a *planning graph* that encodes constraints on possible plans
- Use graph to constrain search for a valid plan
- Planning graph can be built for each problem in a relatively short time
- Extract a solution from planning graph

Planning graph

- Directed, leveled graph with alternating layers of nodes
- Odd layers (*state levels*) represent candidate propositions that could possibly hold at step *i*
- Even layers (*action levels*) represent candidate actions that could possibly be executed at step *i*, including maintenance actions [do nothing]
- **Arcs** represent *preconditions*, *adds* and *deletes*
- Can only execute one real action at a step, but the data structure keeps track of all actions & states that are *possible*

GraphPlan properties

- STRIPS operators: conjunctive preconditions, no conditional or universal effects, no negations
  - Planning problem must be convertible to propositional representation
  - NO continuous variables, temporal constraints, …
  - Problem size grows exponentially
- Finds "shortest" plans (by some definition)
- Sound, complete, and will terminate with failure if there is no plan

Having your cake & eating it too

Init(Have(Cake) $\wedge$ $\neg$Eaten(Cake))

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

Action(Eat(Cake)

PRECOND: Have(Cake)

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

Action(Bake(Cake)

PRECOND: $\neg$Have(Cake)

EFFECT: Have(Cake)

What actions and what literals?

- Add an action in level A_i if ***all*** of its preconditions are present in level S_i
- Add a literal in level S_i if it is the effect of ***some*** action in level A_{i-1} (*including no-ops*)
- Level S_0 has all of the literals from initial state

Planning Graph for Cake Example

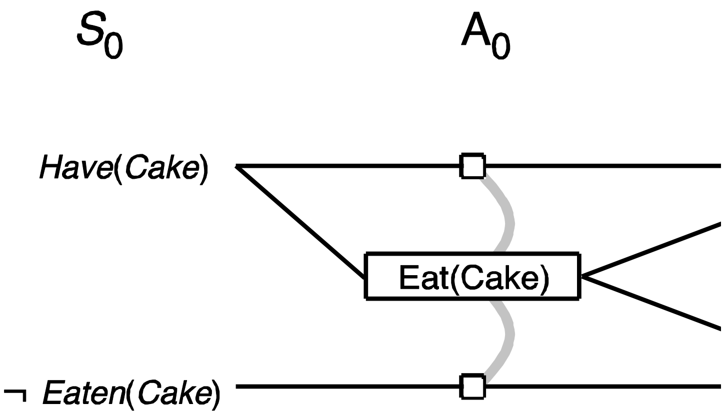
S_0

$Have(Cake)$

$\neg\ Eaten(Cake)$

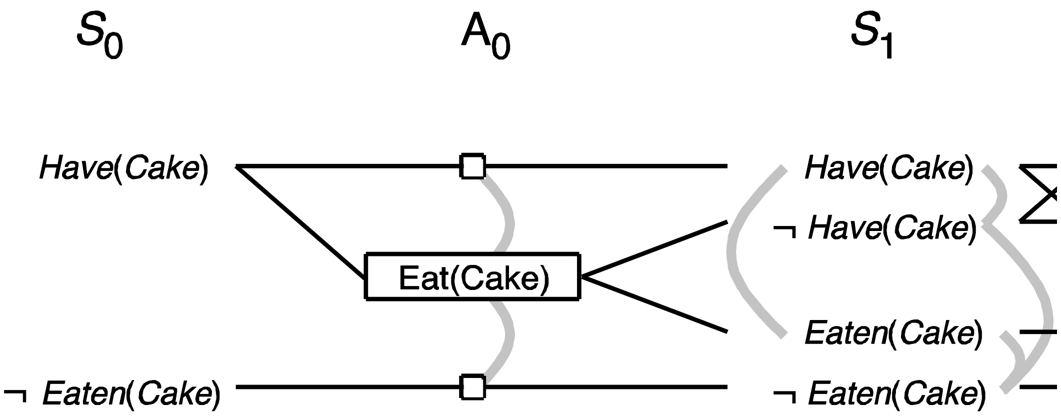
- Level S_0 has all literals from initial state

Planning Graph for Cake Example



- Level S_0 has all literals from initial state
- **Level A_0 has all actions whose preconditions are satisfied in S_0, including no-ops**

Planning Graph for Cake Example



- Level S_0 has all literals from initial state
- Level A_0 has all actions whose preconditions are satisfied in S_0, including no-ops
- **Actions connect preconditions to effects**
- **Gray arcs connect propositions that are mutex (mutually exclusive) & actions that are mutex**

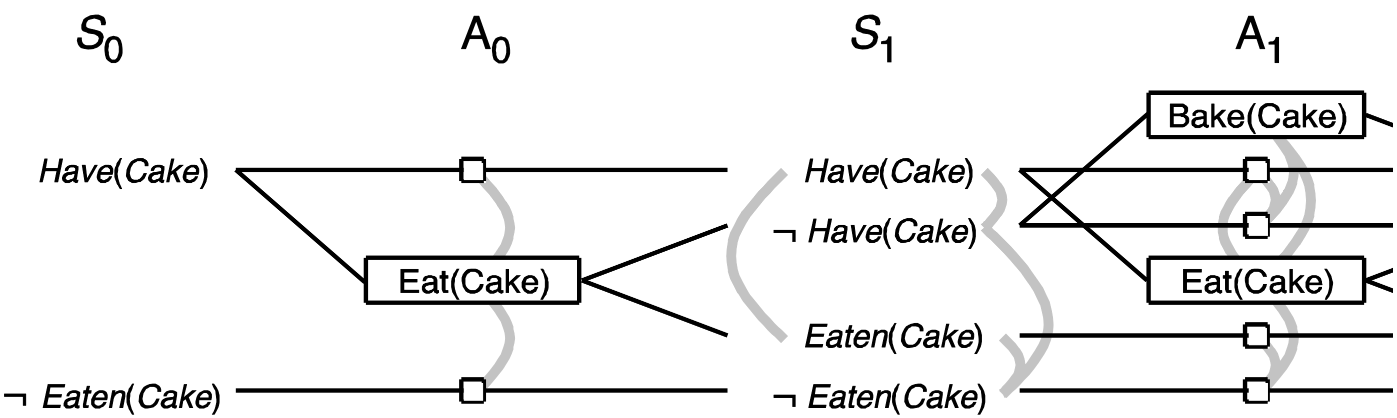
Mutex Arcs

- Mutex arc between two actions indicates that it is impossible to perform the actions in parallel
- Mutex arc between two literals indicates that it is impossible to have these both true at this stage

Computing mutexes

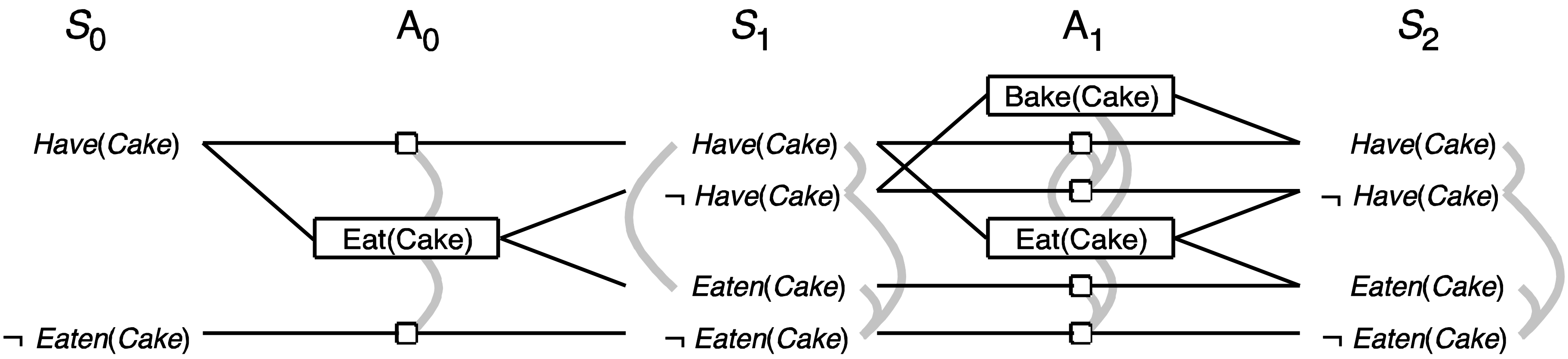
- Mutex actions
  - Inconsistent effects: two actions that lead to inconsistent effects
  - Interference: an effect of first action negates precondition of other action
  - Competing needs: a precondition of first action is mutex with a precondition of second action
- Mutex literals
  - one literal is negation of the other one
  - Inconsistency support: each pair of actions achieving the two literals are mutually exclusive

Planning Graph for Cake Example



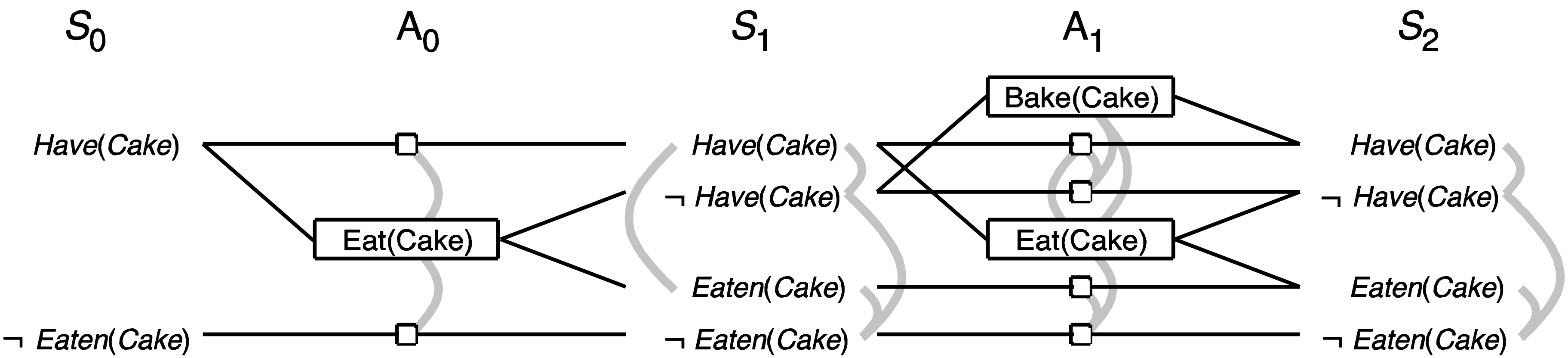
- Actions connect preconditions to effects
- Gray arcs connect propositions that are mutex
- **Actions at level A_i must have support from a set of literals in state S_i that have no mutex relations among themselves**

Planning Graph for Cake Example



- Actions at level A_i must have support from a set of literals in state S_i that have no mutex relations among themselves
- **Stop when the set of literals does has not changed**

Planning Graph for Cake Example



- If all of the literals in the goal are in the final state and are non-mutex …
- We can try to extract a plan from the plan graph

GraphPlan

function GRAPHPLAN(*problem*) **returns** solution or failure

graph ← INITIAL-PLANNING-GRAPH(*problem*)
goals ← CONJUNCTS(*problem*.GOAL)
nogoods ← an empty hash table
for t = 0 **to** ∞ **do**
if *goals* all non-mutex in S_t of *graph* **then**
solution ← EXTRACT-SOLUTION(*graph, goals,* NUMLEVELS(*graph*), *nogoods*)
if *graph* and *nogoods* have both leveled off **then return** *failure*
graph ← EXPAND-GRAPH(*graph*, *problem*)

From Fig. 10.9, p. 383

Spare Tire Problem

Init(Tire(Flat) $\wedge$ Tire(Spare) $\wedge$ At(Flat,Axle) $\wedge$ At(Spare,Trunk))

Goal(At(Spare,Axle))

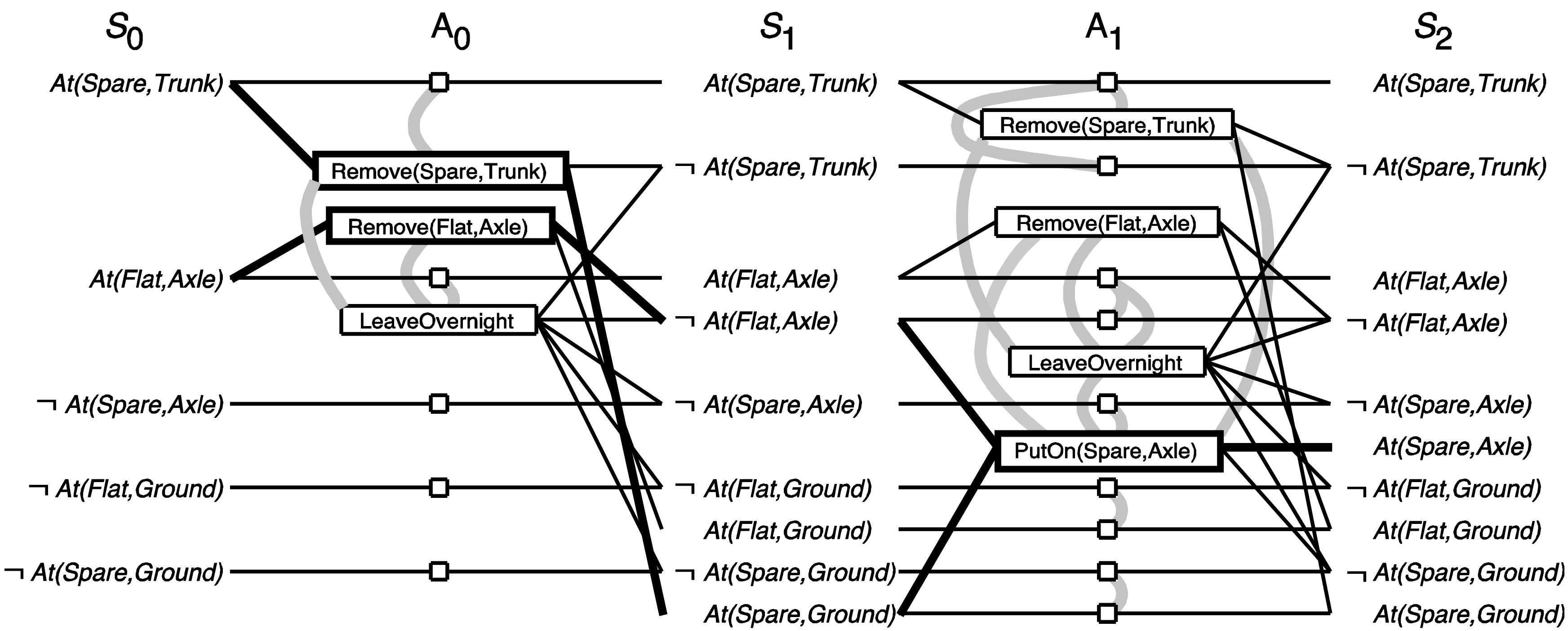
Action(Remove(obj,loc),
    PRECOND: At(obj,loc),
    EFFECT: $\neg$At(obj,loc) $\wedge$ At(obj,Ground))

Action(PutOn(t, Axle),
    PRECOND: Tire(t) $\wedge$ At(t,Ground) $\wedge$ $\neg$At(Flat,Axle),
    EFFECT: $\neg$At(t,Ground) $\wedge$ At(t,Axle))

Action(LeaveOvernight,
    PRECOND: $\emptyset$,
    EFFECT: $\neg$At(Spare,Ground) $\wedge$ $\neg$At(Spare,Axle) $\wedge$ $\neg$At(Spare,Trunk) $\wedge$
    $\neg$At(Flat,Ground) $\wedge$ $\neg$At(Flat,Axle) $\wedge$ $\neg$At(Flat,Trunk))

From Fig. 10.2, p. 370

Spare Tire Planning Graph



From Fig. 10.10, p. 384

Planning graph for heuristic search

- Using the planning graph to estimate the number of actions to reach a goal

- If a literal does not appear in the final level of the planning graph, then there is no plan that achieve this literal!
  - $h = \infty$

Heuristics

- **max-level:** take the maximum level where any literal of the goal first appears
  - admissible
- **level-sum:** take the sum of the levels where any literal of the goal first appears
  - not admissible, but generally efficient (specially for independent subplans)
- **set-level:** take the minimum level where all the literals of the goal appear and are free of mutex
  - admissible

BlackBox Planner

STRIPS-based plan representation

↓

Planning graph

↓

CNF representation

↓

CSP/SAT solver

↓

CSP solution

↓

Plan