UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

AI – Planning

10/29

Cindi Thompson

What is planning?

- Generate sequences of actions to perform tasks and achieve objectives
  - States, actions and goals
- Search for solution over abstract space of plans.
- Classical planning environment: fully observable, deterministic, finite, static and discrete.
- Assists humans in practical applications
  - design and manufacturing
  - military operations
  - games
  - space exploration

More Background

Relaxing the above restrictions requires dealing with uncertainty

- Problem types: sensorless, contingency, exploration

Planning 'communities' in AI

Logic-based: Reasoning About Actions & Change

Less formal representations: Classical AI Planning

Uncertainty (UAI): Graphical Models such as

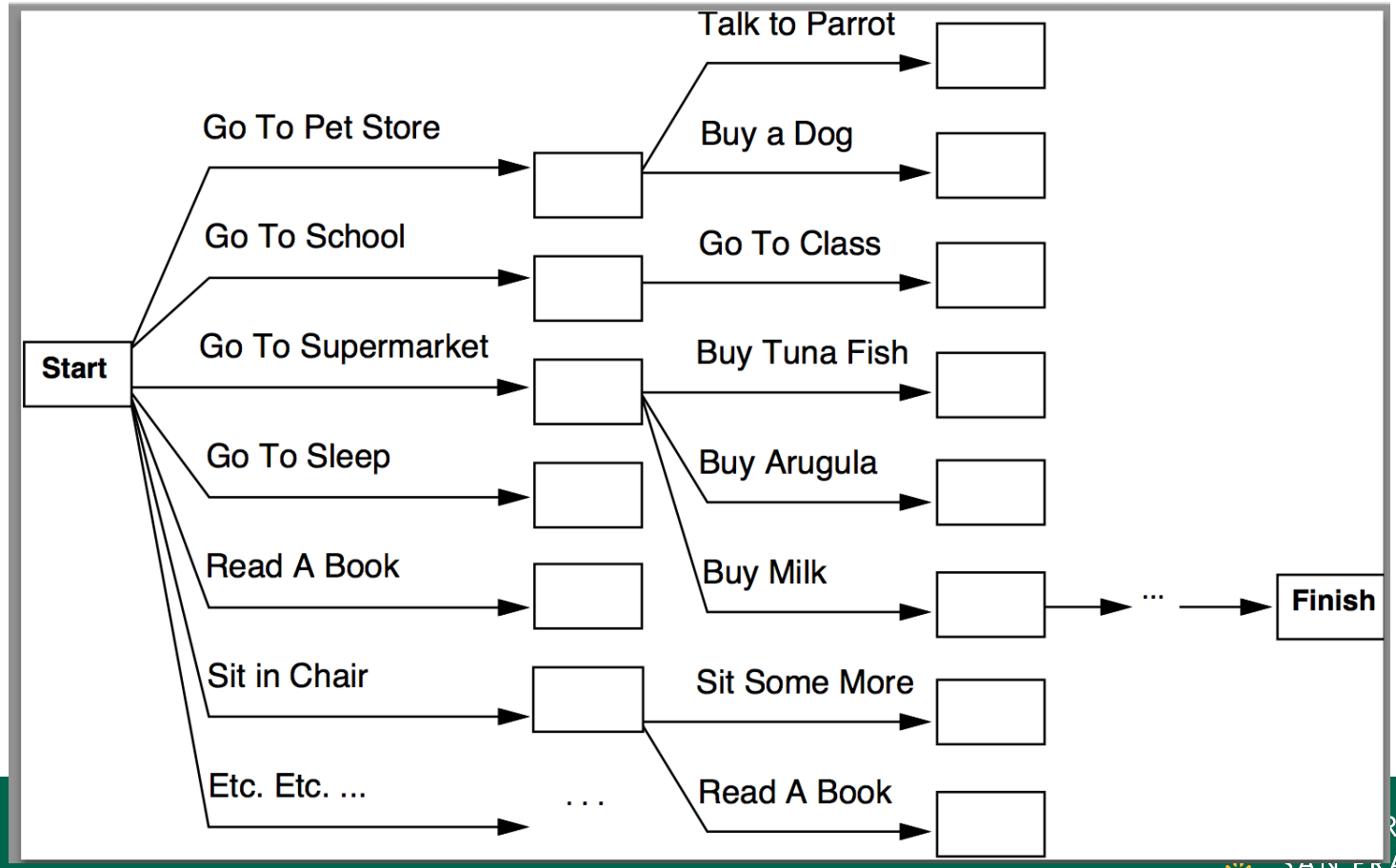
- Markov Decision Processes (MDP), Partially Observable MDPs, etc.

AI Planning is not MRP (Material Requirements Planning)

Why not standard search?

Consider the task: get milk, bananas and a cordless drill

Standard search algorithms fail



Difficulty of Real-World problems

Assume a problem-solving agent using some search method ...

- Which actions are relevant?
  - Exhaustive search vs. backward search
- What is a good heuristic function?
  - Good estimate of the cost of the state?
  - Problem-dependent vs, -independent
- How to decompose the problem?
  - Most real-world problems are ***nearly*** decomposable

Overview

Planning Languages

Searching for a plan

Planning Graphs and GRAPHPLAN

Planning Languages

Languages must represent

States

Goals

Actions

Languages must be

Expressive for ease of representation

Restrictive for efficiency of planning algorithms

Flexible for manipulation by algorithms

STRIPS (old news) and PDDL (Planning Domain Defn. Lang.)

State Representation

- A state is represented with a conjunction of positive literals
- Using
  - Logical Propositions: Poor ∧ Unknown
  - FOL literals: At(Plane1,SFO) ∧ At(Plan2,JFK)
- FOL literals must be ground & function-free
  - Not allowed: At(x,y) or At(Father(Fred),Sydney)
- Closed World Assumption
  - What is not included is assumed false

Planning Language Features

Representation of goals

- Partially specified state and represented as a conjunction of positive ground literals
- A goal is satisfied if the state contains all literals in goal.
- Example:
  - Rich ∧ Famous ∧ Miserable

    satisfies the goal

    Rich ∧ Famous

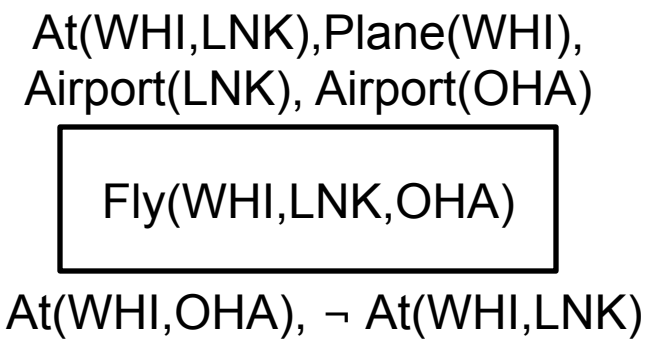
Action Representation

Action Schema

Action name

Preconditions

Effects



Example

Action(Fly(p,from,to),

PRECOND: At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

EFFECT: ¬At(p,from) ∧ At(p,to))

Sometimes, Effects are split into ADD list and DELETE list

Semantics: Applying an Action

- An action is applicable in any state that satisfies the precondition
- Find a substitution list θ for the variables of all the precondition literals

Example:

At(P1,JFK) $\wedge$ *At(P2,SFO)* $\wedge$ *Plane(P1)* $\wedge$ *Plane(P2)* $\wedge$ *Airport(JFK)* $\wedge$ *Airport(SFO)*

Satisfies *: At(p,from)* $\wedge$ *Plane(p)* $\wedge$ *Airport(from)* $\wedge$ *Airport(to)*

With θ ={ *p/P1, from/JFK, to/SFO*}

Thus the action is applicable.

Semantics: Applying an Action, continued

The result of executing action a in state s is the state s'

- s' is same as s except:
  - Any positive literal *P* in the EFFECT of *a* is added to *s*'
  - Any negative literal ¬*P* is removed from *s*'

Example:

s: At(P1,JFK) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

EFFECT of action Fly(P1,JFK,SFO): ¬*AT(p,from)* ∧ *At(p,to)*

The new state s':

At(P1,SFO) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

NOTE: every literal NOT in the effect remains unchanged

Example: air cargo transport

Init(At(C1, SFO) ∧ At(C2,JFK) ∧ At(P1,SFO) ∧ At(P2,JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1,JFK) ∧ At(C2,SFO))

Action(Load(c,p,a)
PRECOND: *At(c,a) ∧ At(p,a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)*
EFFECT: ¬*At(c,a) ∧ In(c,p))*
Action(Unload(c,p,a)
PRECOND: *In(c,p) ∧ At(p,a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)*
EFFECT: *At(c,a) ∧* ¬*In(c,p))*
Action(Fly(p,from,to)
PRECOND: *At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)*
EFFECT: ¬ *At(p,from) ∧ At(p,to))*

[Load(C1,P1,SFO), Fly(P1,SFO,JFK), Unload(C1, P1, JFK), Load(C2,P2,JFK), Fly(P2,JFK,SFO), Unload(C2, P2, SFO]

Example: Spare tire problem

Init(At(Flat, Axle) ∧ At(Spare,Trunk))
Goal(At(Spare,Axle))

Action(Remove(Spare,Trunk)
PRECOND: *At(Spare,Trunk)*
EFFECT: ¬*At(Spare,Trunk)* ∧ *At(Spare,Ground))*
Action(Remove(Flat,Axle)
PRECOND: *At(Flat,Axle)*
EFFECT: ¬*At(Flat,Axle)* ∧ *At(Flat,Ground))*
Action(PutOn(Spare,Axle)
PRECOND: *At(Spare,Groundp)* ∧¬*At(Flat,Axle)*
EFFECT: *At(Spare,Axle)* ∧ ¬*At(Spare,Ground))*
Action(LeaveOvernight
PRECOND:
EFFECT: ¬ *At(Spare,Ground)* ∧ ¬ *At(Spare,Axle)* ∧ ¬ *At(Spare,trunk)*
∧ ¬ *At(Flat,Ground)* ∧ ¬ *At(Flat,Axle))*

Example: Blocks world

Init(On(A, Table) ∧ On(B,Table) ∧ On(C,Table) ∧ Block(A) ∧
Block(B) ∧ Block(C) ∧ Clear(A) ∧ Clear(B) ∧ Clear(C))

Goal(On(A,B) ∧ On(B,C))

Action(Move(b,x,y)
PRECOND: *On(b,x) ∧ Clear(b) ∧ Clear(y) ∧ Block(b) ∧ (b ≠ x)*
∧ (b ≠ y) ∧ (x ≠ y)
EFFECT: *On(b,y) ∧ Clear(x) ∧ ¬ On(b,x) ∧ ¬ Clear(y))*

Action(MoveToTable(b,x)
PRECOND: *On(b,x) ∧ Clear(b) ∧ Block(b) ∧ (b≠ x)*
EFFECT: *On(b,Table) ∧ Clear(x) ∧ ¬ On(b,x))*

State-Space Search (1)

Search the space of states (first chapters)

Initial state, goal test, step cost, etc.

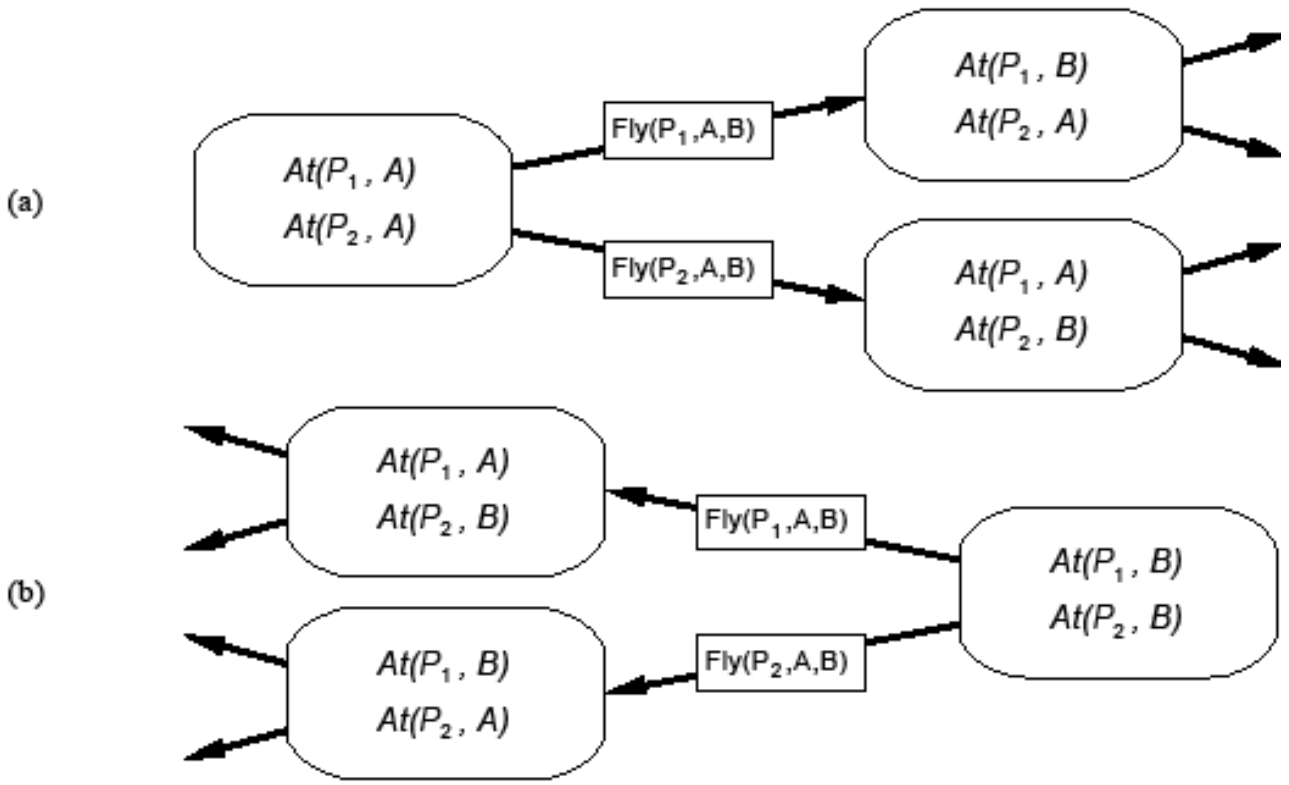
Actions are the transitions between state

Actions are invertible (why?)

Move forward from the initial state: Forward State-Space Search or Progression Planning

Move backward from goal state: Backward State-Space Search or Regression Planning

State-Space Search (2)



Progression algorithm

Formulation as state-space search problem:

- Initial state = initial state of the planning problem
  - Literals not appearing are false
- Actions = those whose preconditions are satisfied
  - Add positive effects, delete negative
- Goal test = does the state satisfy the goal
- Step cost = each action costs 1

No function symbols ... any graph search that is complete is a complete planning algorithm.

– E.g. A*

Inefficient:

– (1) irrelevant action problem

– (2) good heuristic required for efficient search

Regression algorithm

What are the predecessors of a state? Need to be able to apply some action to lead to the current goal state.

Example from cargo domain:

- Goal state = *At(C1, B) ∧ At(C2, B) ∧ ... ∧ At(C20, B)*
- Relevant action for first conjunct*: Unload(C1,p,B)*
- Works only if pre-conditions of relevant action are satisfied.
- Previous state = *In(C1, p) ∧ At(p, B) ∧ At(C2, B) ∧ ... ∧ At(C20, B)*
- Subgoal At(C1,B) should *not* be present in this state.

Consistent action

- The purpose of applying an action is to achieve a desired literal
- We should be careful that the action does not undo a (different) desired literal (as a side effect)
- A consistent action is an action that does not undo a desired literal

Must check this in regression search

Advantage of regression search:

– Often much lower branching factor than forward search.

Regression algorithm

Given

A goal *G* description

An action *A* that is relevant and consistent

Generate a predecessor state where

Positive effects (literals) of *A* in *G* are deleted

Precondition literals of *A* are added unless they already appear

Substituting any variables in A's effects to match literals in G

Substituting any variables in A's preconditions to match substitutions in A's effects

Repeat until predecessor description matches initial state

Search in the regression algorithm

Still need to find the “goal” (initial state). NP-hard!
To apply A*, need an admissible heuristic. Approaches:

- Find the optimal solution to a relaxed problem
  - Remove all preconditions from actions
  - Ignore delete lists
- The subgoal independence assumption:
  - The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Partial-order planning

Progression and regression planning are *totally ordered plan search* forms.

– They cannot take advantage of problem decomposition.

– Decisions must be made on how to sequence actions on all the subproblems

Least commitment strategy:

– Delay choice during search

Any planning algorithm that can place two actions into a plan without deciding which comes first is a Partial-order (PO) planner.

Shoe example

Goal(RightShoeOn $\wedge$ LeftShoeOn)

Init()

Action(RightShoe, PRECOND: RightSockOn EFFECT:
RightShoeOn)

Action(RightSock, PRECOND: EFFECT: RightSockOn)

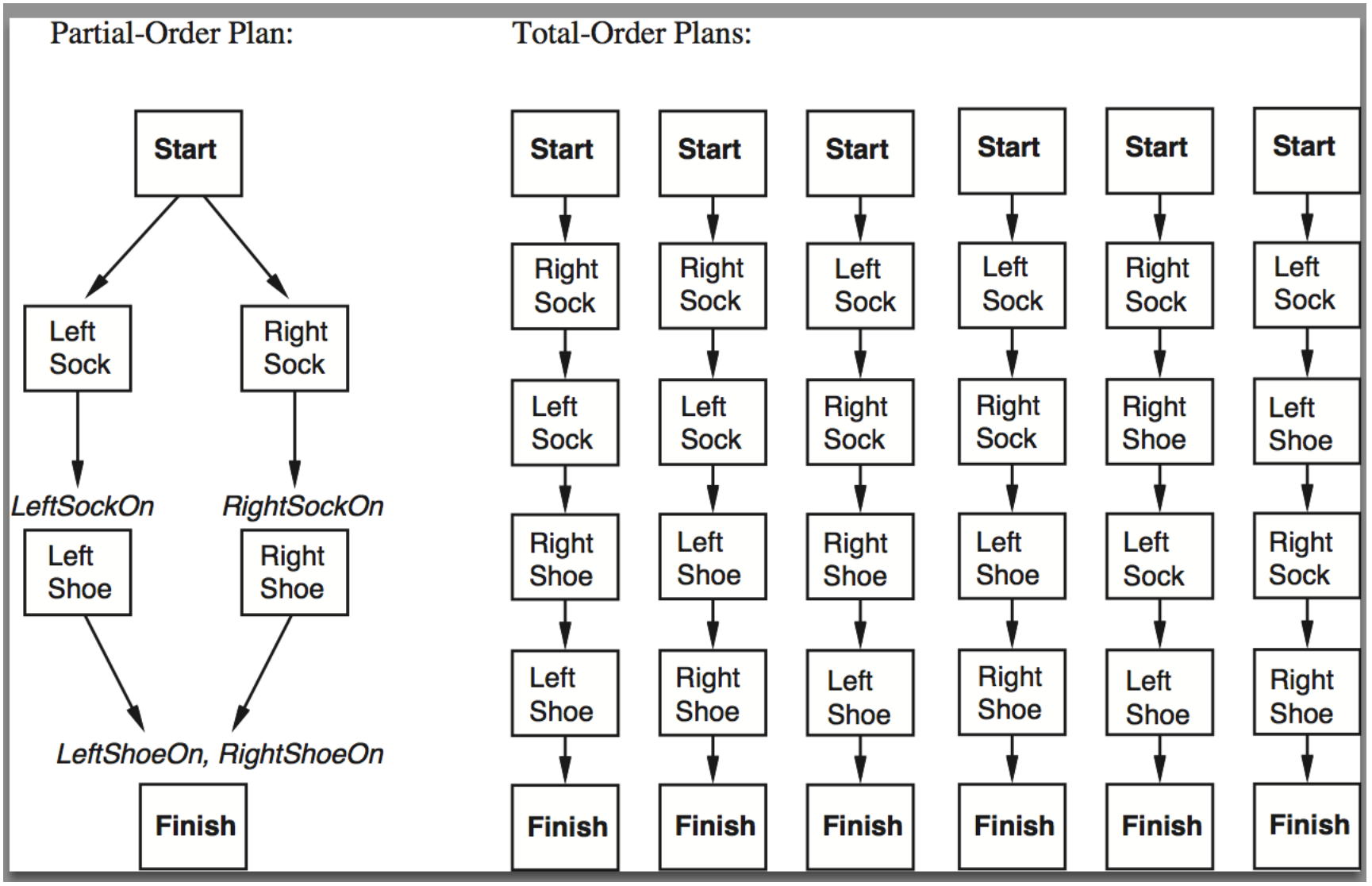
Action(LeftShoe, PRECOND: LeftSockOn EFFECT: LeftShoeOn)

Action(LeftSock, PRECOND: EFFECT: LeftSockOn)

Planner: combine two action sequences

(1) leftsock, leftshoe (2) rightsock, rightshoe

POP and its corresponding ordered plans



Planning Graphs

- How can we achieve better heuristic estimates?
- Data structure: planning graph
  - Has an associated solution extraction algorithm, GRAPHPLAN
- Sequence $\langle S_0, A_0, S_1, A_1, \ldots, S_i \rangle$ of levels
  - Alternating state levels & action levels
  - Levels correspond to time stamps in the plan
  - Starting at initial state
  - State level is a set of (propositional) literals
    - All literals that *could* be true at that level
  - Action level is a set of (propostionalized) actions
    - All those actions that *could* have their preconditions satisfied
- Propositionalization may yield combinatorial explosion in the presence of a large number of objects

Planning Graphs: Could?

State level is a set of (propositional) literals

- All literals that *could* be true at that level: depending on the actions actually executed at the preceding time step

Action level is a set of (propostionalized) actions

- All those actions that *could* have their preconditions satisfied, depending on which of the literals actually hold

Records only a restricted subset of possible negative interactions among actions

Think of it as a polynomial approximation to the fully instantiated search tree

Can estimate number of steps needed, and is admissible

Mid-class overview

Building the Planning Graph

Using it for Heuristic Estimation

Using it for generating the plan

Example

Init(Have(Cake))

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

Action(Eat(Cake), PRECOND: Have(Cake)
EFFECT: ¬Have(Cake) $\wedge$ Eaten(Cake))

Action(Bake(Cake), PRECOND: ¬ Have(Cake)
EFFECT: Have(Cake))

Example

Init(Have(Cake))

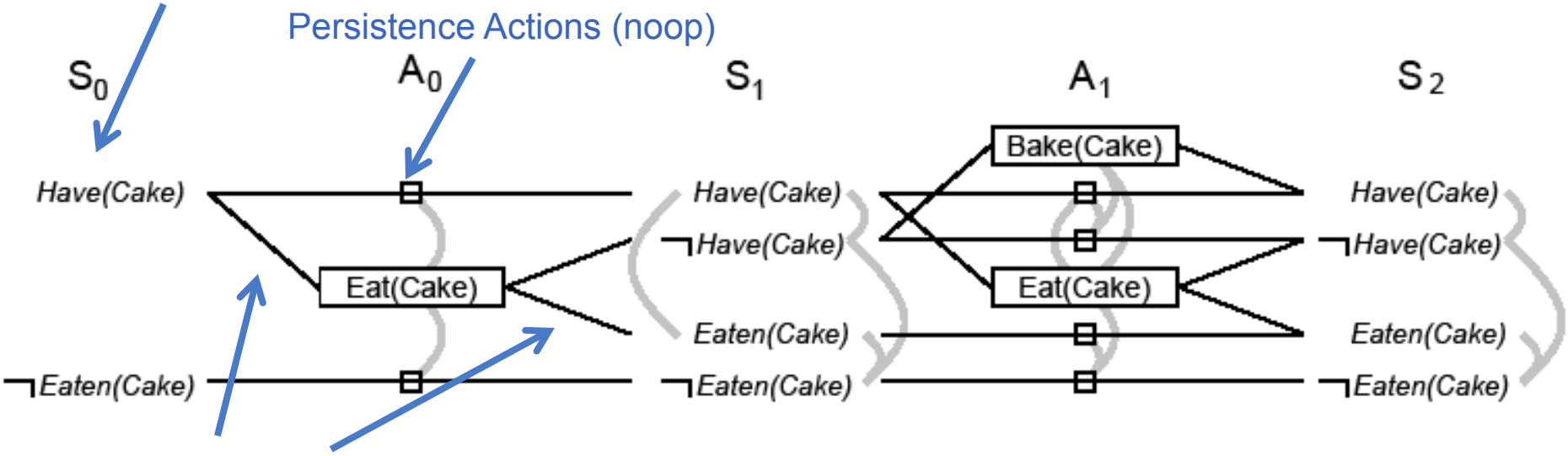
Goal(Have(Cake) ∧ Eaten(Cake))

Action(Eat(Cake), PRECOND: Have(Cake)
EFFECT: ¬Have(Cake) ∧ Eaten(Cake))

Action(Bake(Cake), PRECOND: ¬ Have(Cake)
EFFECT: Have(Cake))

Propositions true at the initial state

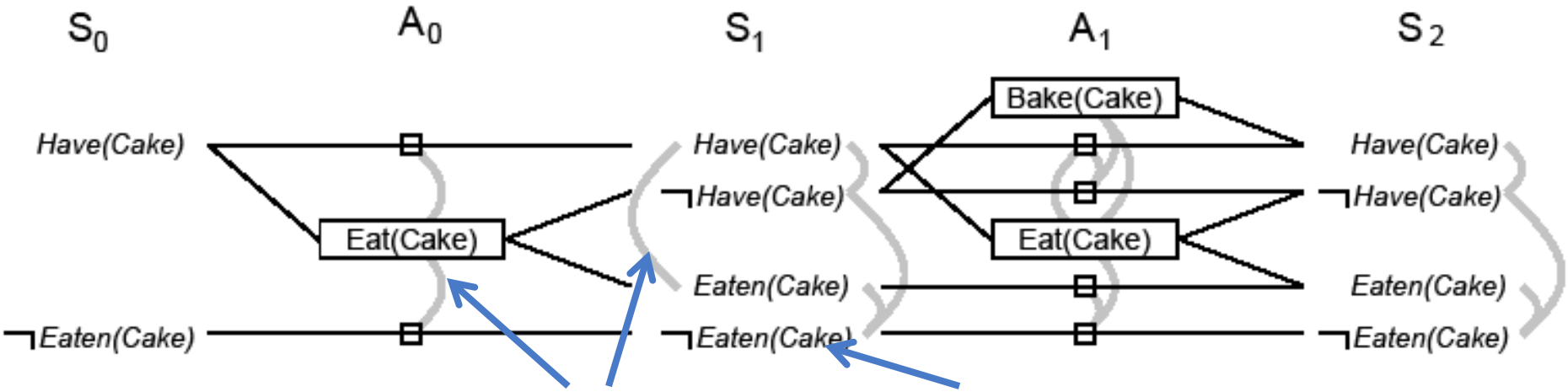
Persistence Actions (noop)



Action is connected to its preconds & effects

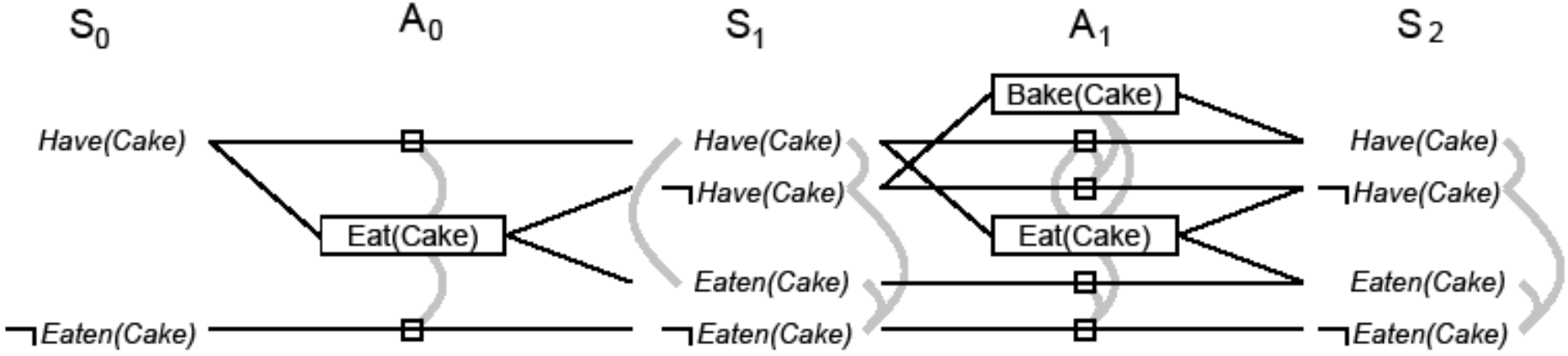
Example, continued

- At each state level, list all literals that may hold at that level
- At each action level, list all noops & all actions whose preconditions may hold at previous levels
- Repeat until plan 'levels off,' no new literals appears ($S_i = S_{i+1}$)
- Building the Planning Graph is a polynomial time process
- Add (binary) mutual exclusion (mutex) links between conflicting actions and between conflicting literals



Mutex links between actions

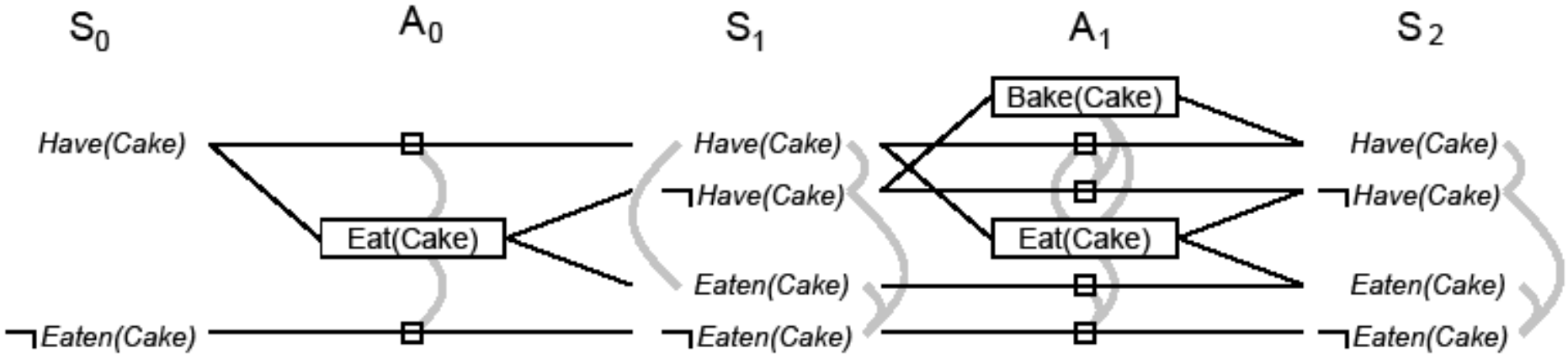
1. **Inconsistent effects:** one action negates an effect of another
- Eat(Cake) & noop of Have(Cake) disagree on effect Have(Cake)
2. **Interference:** An action effect negates the precondition of another
- Eat(Cake) negates precondition of the noop of Have(Cake):
3. **Competing needs:** A precondition on an action is mutex with the precondition of another
- Bake(Cake) & Eat(Cake): compete on Have(Cake) precondition



Mutex links between literals

1. Two literals are negation of each other
2. **Inconsistent support**: Each pair of actions that can achieve the two literals is mutex. Examples:

- In S1, Have(Cake) & Eaten(Cake) are mutex
- In S2, they are not because Bake(Cake) & the noop of Eaten(Cake) are not mutex



Planning Graphs for heuristic estimation

- A literal that does not appear in the final level cannot be achieved by any plan
  - State-space search: Any state containing an unachievable literal has cost h(n) = ∞
  - POP: Any plan with an unachievable open condition has cost h(n)=∞
- The estimated cost of any goal literal is the first level at which it appears
  - Estimate is admissible for individual literals
  - Estimate can be improved by serializing the graph (serial planning graph: one action per level) by adding mutex between all actions in a given level
- The estimate of a conjunction of goal literals
  - Three heuristics: max level, level sum, set level

Estimate of Conjunction of Goal Literals

- Max-level
  - The largest level of a literal in the conjunction
  - Admissible, not very accurate
- Level sum
  - Under subgoal independence assumption, sums the level costs of the literals
  - Inadmissible, works well for largely decomposable problems
- Set level
  - Finds the level at which all literals appear w/o any pair of them being mutex
  - Dominates max-level, works extremely well on problems where there is a great deal of interaction among subplans

GRAPHPLAN Algorithm

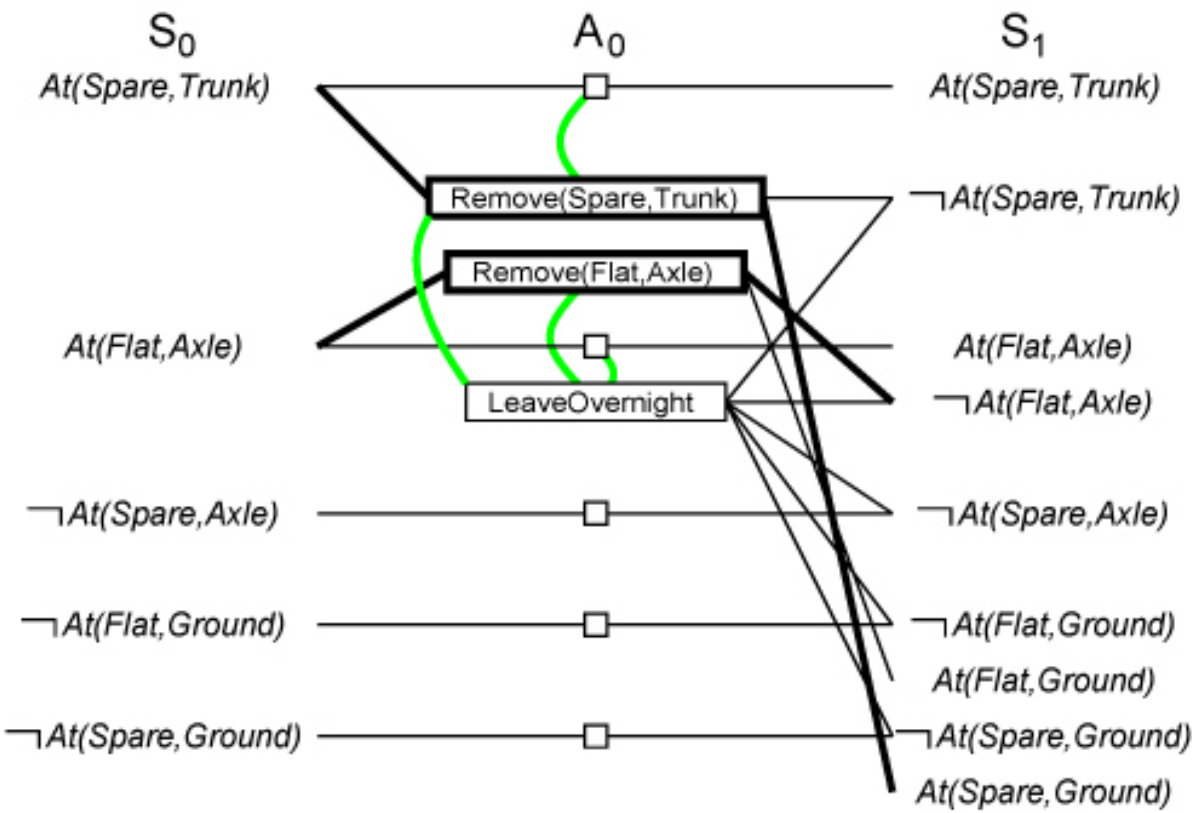
Extracting a solution from the planning graph

GRAPHPLAN(*problem*) **returns** *solution* or *failure*
graph ← INITIALPLANNINGGRAPH(*problem*)
goals ← GOALS[*problem*]
loop do
  if *goals* all non-mutex in last level of graph **then do**
    solution ← EXTRACTSOLUTION(*graph, goals,* LENGTH(*graph*))
    if *solution* ≠ *failure* **then return** *solution*
    else if NOSOLUTIONPOSSIBLE(*graph)* **then return** *failure*
  graph ← EXPANDGRAPH(*graph*, *problem*)

GRAPHPLAN Algorithm – Example

- At(Spare,Axle) is not in S_0
- Can't yet extract solution
- Expand the plan

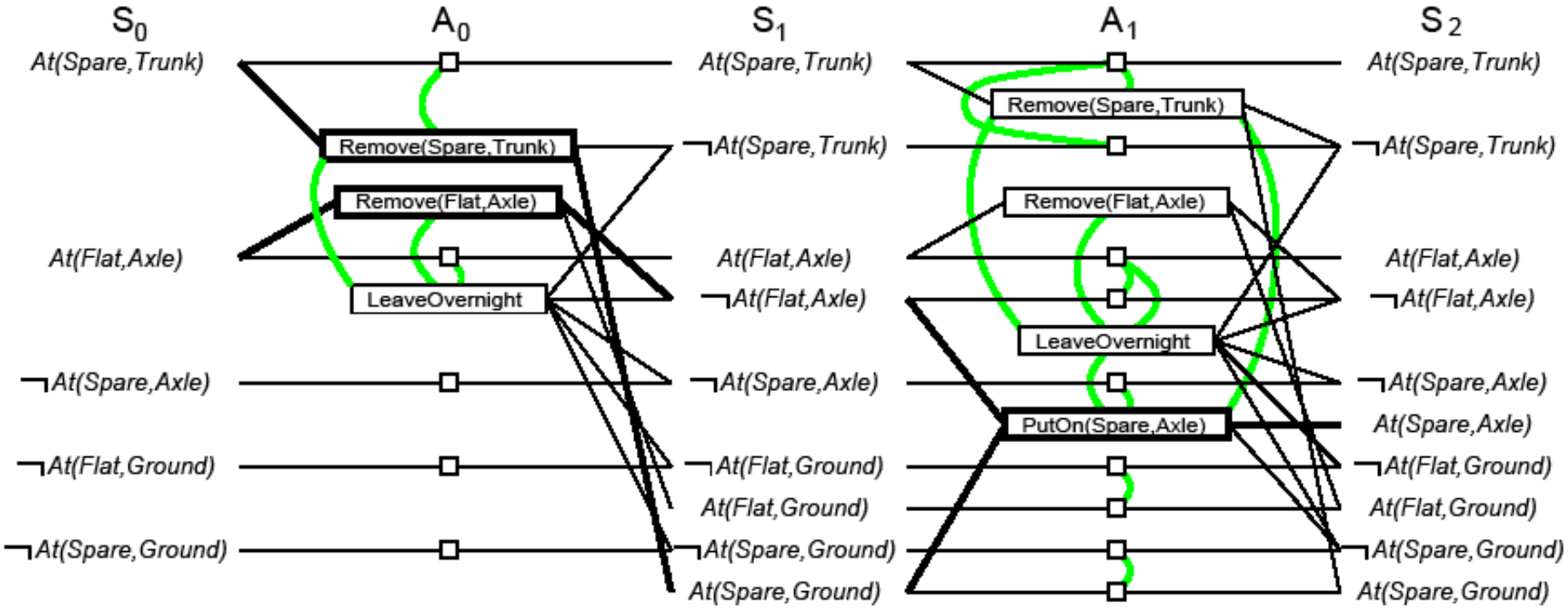
GRAPHPLAN Algorithm – Example, cont'd



- Three actions are applicable
- 3 actions and 5 noops are added
- Mutex links are added
- At(Spare,Axle) still not in S_1
- Plan is expanded

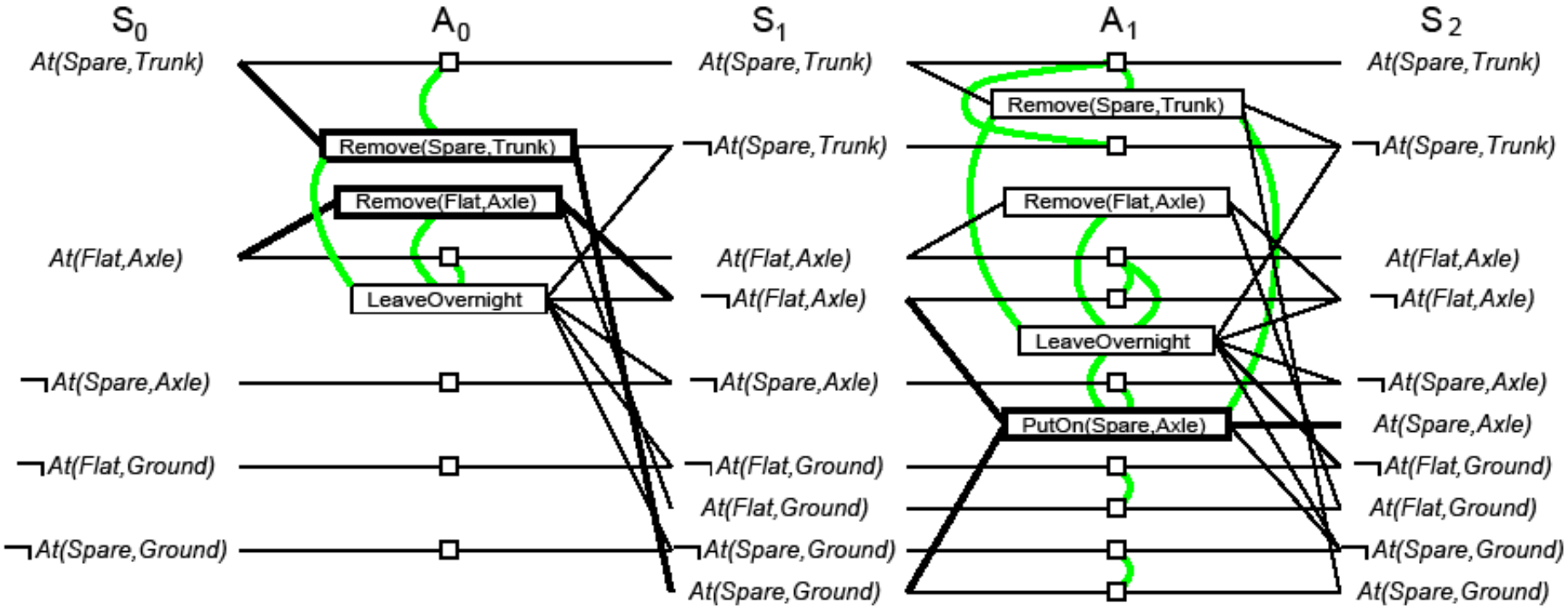
GRAPHPLAN Algorithm – Example, cont'd

Illustrates well mutex links: inconsistent effects, interference, competing needs, inconsistent support



GRAPHPLAN Algorithm – Solution extraction

The goal literals exist in S2 and are not mutex with any other. Time to apply EXTRACT-SOLUTION



GRAPHPLAN Algorithm – Solution extraction

EXTRACT-SOLUTION two approaches

1. Use Boolean CSP: Variables are actions, domains are {0=out of plan, 1=in plan}, constraints are the mutex links
2. Apply search!

– Initial state = last level of PG and goals of planning problem

– Actions = select any set of non-conflicting actions that cover the goals in the state

– Goal = reach level S0 such that all goals are satisfied

– Cost = 1 for each action

Termination of GRAPHPLAN

- GRAPHPLAN is guaranteed to terminate
  - Literal increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotinically
- A solution is guaranteed not to exist when
  - The graph levels off with all goals present & non-mutex, and
  - EXTRACTSOLUTION fails to find solution

Optimality of GRAPHPLAN

- The plans generated by GRAPHPLAN
  - Are optimal in the number of steps needed to execute the plan
  - Not necessarily optimal in the number of actions in the plan (GRAPHPLAN produces partially ordered plans)

Planning vs. Scheduling

- (Classical) planning, as in this lecture:
  - What to do?
  - In what order?
- But not:
  - How long?
  - When?
  - Using what resources?
- Normally:
  - plan first, schedule later

Scheduling Problem Representation

Job shop scheduling:

- A set of jobs
- Each job is a collection of ACTIONS with some ORDERING CONSTRAINTS
- Each action has a DURATION and a set of RESOURCE CONSTRAINTS
- Resources may be CONSUMABLE or REUSABLE

Solution:

- Start times for all actions, obeying all constraints

Summary

- Standard search inadequate for real-world planning scenarios
- PDDL is a general representation for planning problems
- Regression (or backwards) search for plans more efficient than progression search, but requires heuristics
- Planning graphs can provide these heuristics. Discussed:
  - Building the graph
  - Using it for heuristic estimation
  - Using it for generating a plan

Summary

- Applications
  - NASA mission planning
  - Robot planning more generally
  - Recent article on self-assembling chairs and cancer fighting nano-robots: http://www.theguardian.com/artanddesign/architecture-design-blog/2013/apr/10/4d-printing-cancer-nano-robots
  - Future: Can a computer plan a courtroom strategy? Or write your software project plan for you?