DRONACHARYA
College of Engineering

INTELLIGENT SYSTEMS (CSE-303-F)

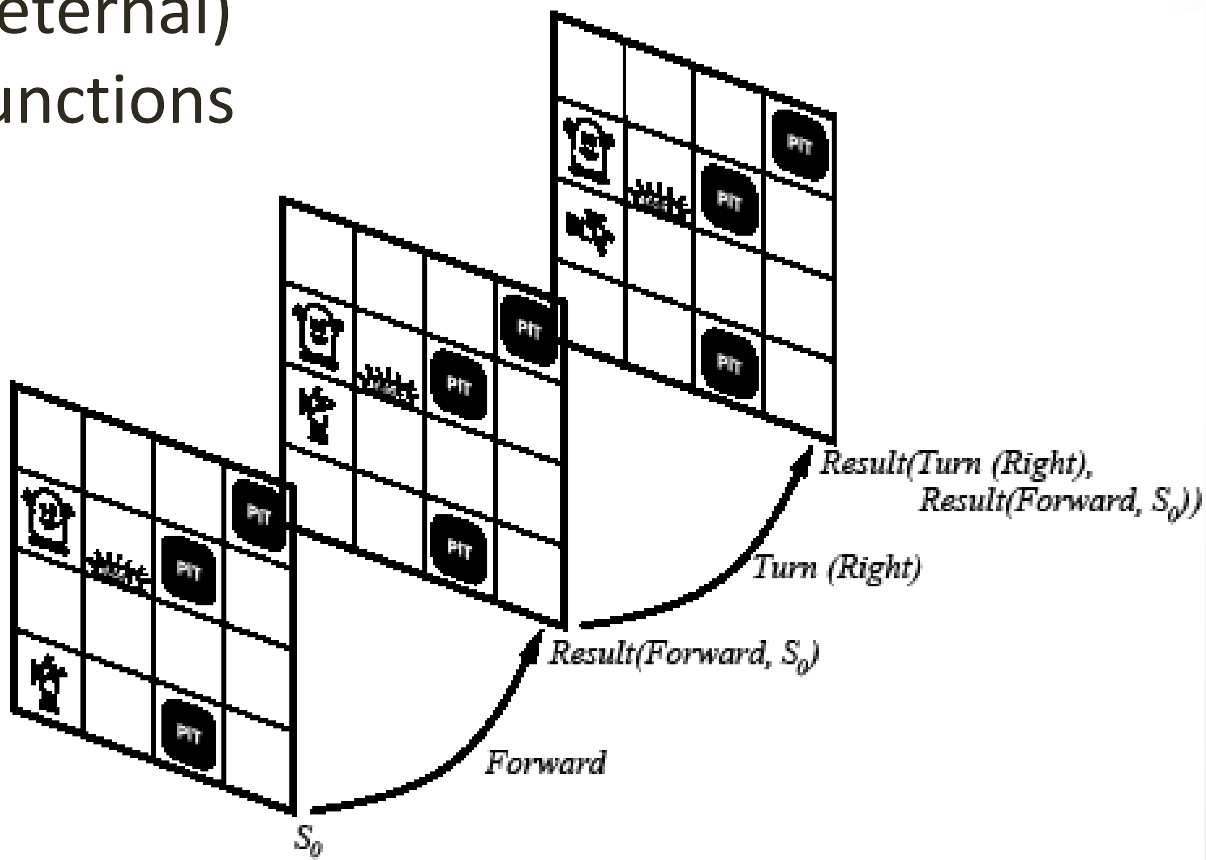
Section C

# Planning

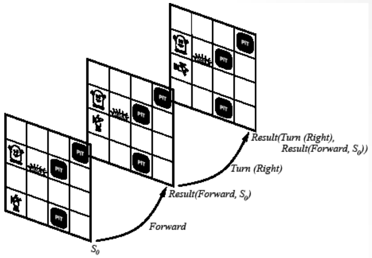
# Situation Calculus: Ontology

- Situations
- Fluents
- Atemporal (or eternal) predicates & functions

***AIMA Section 10.3***



# Situation Calculus: Ontology




- Situations
  - Initial state: $S_0$
  - A function *Result*(*a.s*) gives the situation resulting from applying action *a* in situation *s*
- Fluents
  - Functions & predicates whose truth values can change from one situation to the other
  - Example: $\neg Holding(G_1,S_0)$
- Atemporal (or eternal) predicates and functions
  - Example: *Gold*($G_1$), *LeftLegOf(Wumpus)*

# Situation Calculus

- Sequence of actions
  - Result([],s)=s
  - Result([a|seq],s)=Result(seq,Result(a,s))
- Projection task
  - Deducing the outcome of a sequence of actions
- Planning task
  - Find a sequence of actions that achieves a desired effect

# Example: Wumpus World

- Fluents
  - At(o,p,s), Holding(o,s)
- Agent is in [1,1], gold is in [1,2]
  - $At(Agent,[1,1],S_0) \wedge At(G_1,[1,2],S_0)$
- In $S_0$, we also need to have:
  - $At(o,x,S_0) \Leftrightarrow [(o=Agent) \wedge x=[1,1]] \vee [(o=G_1) \wedge x=[1,2]]$
  - $\neg Holding(o,S_0)$
  - $Gold(G_1) \wedge Adjacent([1,1],[1,2]) \wedge Adjacent([1,2],[1,1])$
- The query is:
  - $\exists$ seq $At(G_1,[1,1],Result(seq,S_0))$
- The answer is
  - $At(G1,[1,1],Result(Go([1,1],[1,2]),Grab(G_1),Go([1,2],[1,1]),S_0))$

# Importance of Situation Calculus

- Historical note
  - Situation Calculus was the first attempt to formalizing planning in FOL
  - Other formalisms include Event Calculus
  - The area of using logic for planning is informally called in the literature "Reasoning About Action & Change"
- Highlighted three important problems
  1. Frame problem
  2. Qualification problem
  3. Ramification problem

# ‘Famous’ Problems

- Frame problem
  - Representing all things that stay the same from one situation to the next
  - Inferential and representational
- Qualification problem
  - Defining the circumstances under which an action is guaranteed to work
  - Example: what if the gold is slippery or nailed down, etc.
- Ramification problem
  - Proliferation of implicit consequences of actions as actions may have secondary consequences
  - Examples: How about the dust on the gold?

# Outline

- Background
  - Situation Calculus
  - Frame, qualification, & ramification problems
- **Representation language**
- Algorithms

# Planning Languages

- Languages must represent..
  - States
  - Goals
  - Actions
- Languages must be
  - Expressive for ease of representation
  - Flexible for manipulation by algorithms

# State Representation

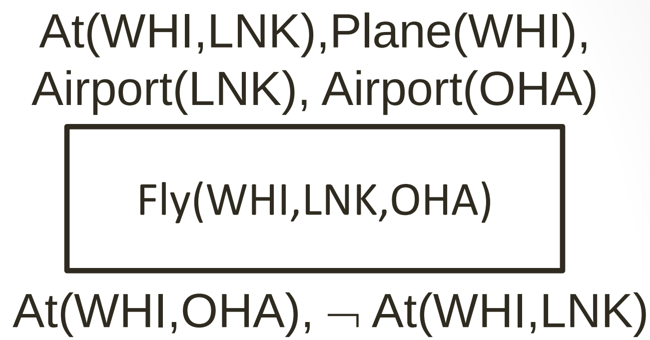
- A state is represented with a conjunction of positive literals
- Using
  - Logical Propositions: $Poor \wedge Unknown$
  - FOL literals: $At(Plane1,OMA) \wedge At(Plan2,JFK)$
- FOL literals must be ground & function-free
  - Not allowed: *At(x,y)* or *At(Father(Fred),Sydney)*
- Closed World Assumption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

# Goal Representation

- Goal is a <u>partially</u> specified state
- A proposition satisfies a goal if it contains all the atoms of the goal and possibly others..
  - Example: Rich $\wedge$ Famous $\wedge$ Miserable satisfies the goal Rich $\wedge$ 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

# Applying an Action

- Find a substitution list $\theta$ for the variables
  - of all the precondition literals
  - with (a subset of) the literals in the current state description
- Apply the substitution to the propositions in the effect list
- Add the result to the current state description to generate the new state
- Example:
  - Current state: At(P1,JFK) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO)
  - It satisfies the precondition with $\theta$={p/P1,from/JFK, to/SFO)
  - Thus the action Fly(P1,JFK,SFO) is applicable
  - The new current state is: At(P1,SFO) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO)

# Languages for Planning Problems

- STRIPS
  - Stanford Research Institute Problem Solver
  - Historically important
- ADL
  - Action Description Languages
  - See Table 11.1 for STRIPS versus ADL
- PDDL
  - Planning Domain Definition Language
  - Revised & enhanced for the needs of the International Planning Competition
  - Currently version 3.1

# Example: Air Cargo

- See Figure 11.2
- Initial state
- Goal State
- Actions: Load, Unload, Fly

# Example: Spare Tire Problem

- See Figure 11.3
- Initial State
- Goal State
- Actions:
  - *Remove(Spare,Trunk), Remove(Flat, Axle)*
  - *PutOn(Spare,Axle)*
  - *LeaveOvernight*
- Note
  - the negated precondition $\neg$*At(Flat,Axle)* not allowed in STRIPS.
  - Could be easily replaced with *Clear(Axle)*, adding one more predicate to the language

# Example: Blocks World

- See Fig 11.4
- Initial state
- Goal
- Actions:
  - *Move*(b,x,y)
  - *MoveToTable*(b,x)

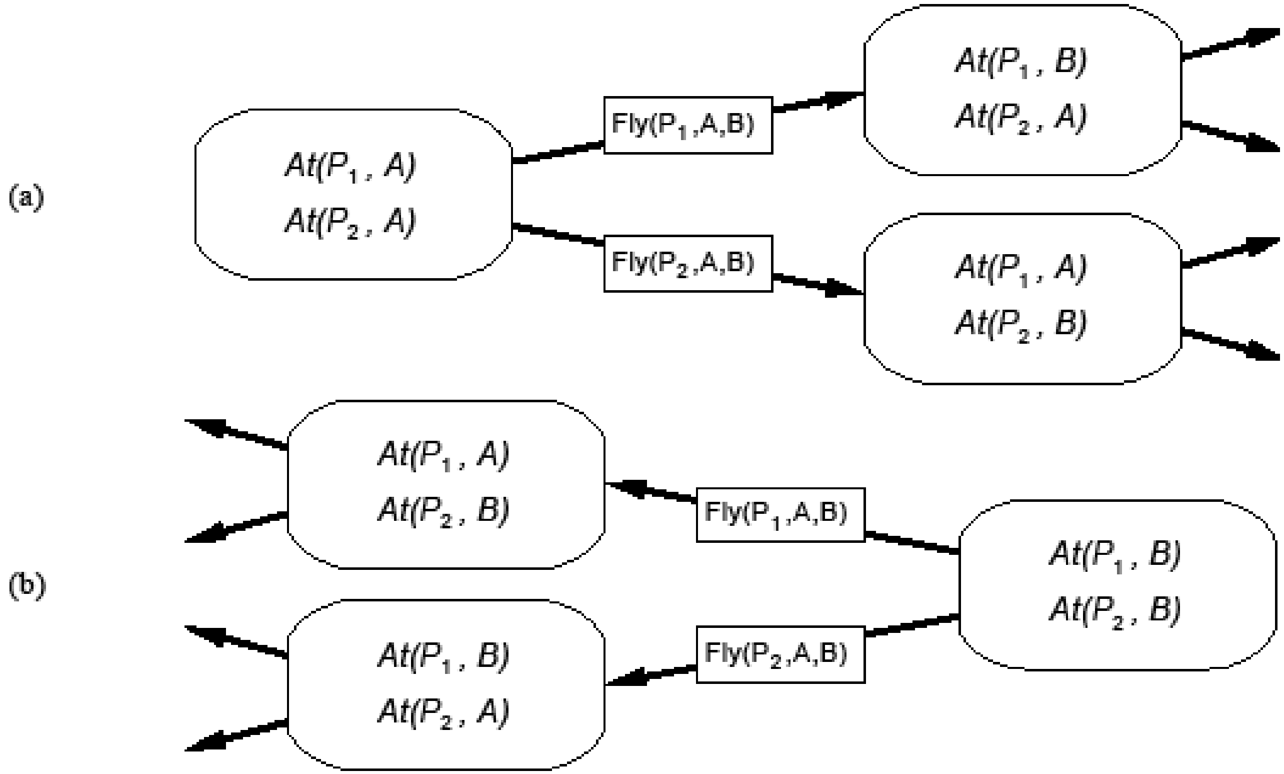
# Outline

- Background
  - Situation Calculus
  - Frame, qualification, & ramification problems
- Representation language
- **Planning Algorithms**
  - **State-Space Search**
  - Partial-Order Planning (POP)
  - Planning Graphs (GRAPHPLAN)
  - SAT Planners

# 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)

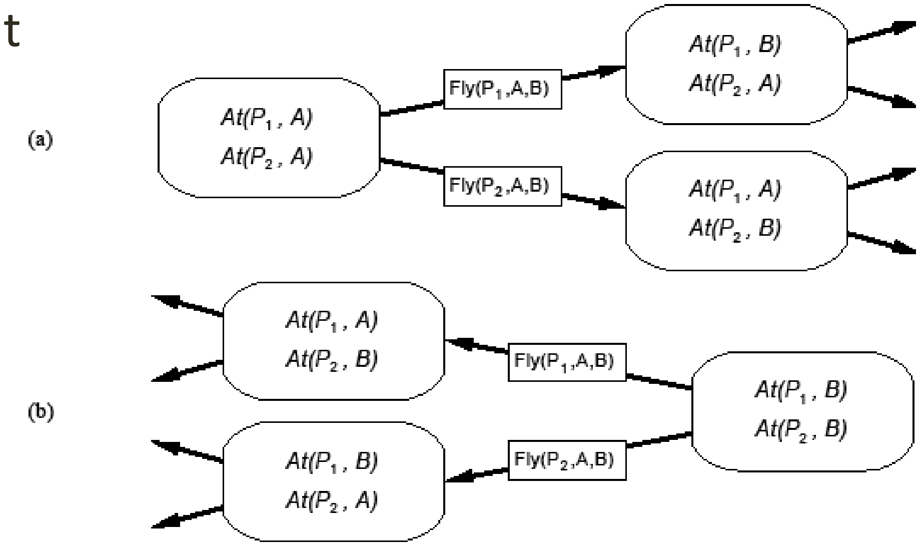


# State-Space Search (3)

- Remember that the language has no functions symbols
- Thus number of states is finite
- And we can use any complete search algorithm (e.g., A*)
  - We need an admissible heuristic
  - The solution is a path, a sequence of actions: total-order planning
- Problem: Space and time complexity
  - STRIPS-style planning is PSPACE-complete unless actions have
    - only positive preconditions and
    - only one literal effect

# SRIPS in State-Space Search

- STRIPS representation makes it easy to focus on 'relevant' propositions and
  - Work backward from goal (using EFFECTS)
  - Work forward from initial state (using PRECONDITIONS)
  - Facilitat



# Relevant Action

- An action is relevant
  - In Progression planning when its preconditions match a subset of the current state
  - In Regression planning, when its effects match a subset of the current goal state

# Consistent Action

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

# Backward State-Space Search

- 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

# Heuristic to Speed up Search

- We can use A*, but we need an admissible heuristic
  1. Divide-and-conquer: sub-goal independence assumption
  - Problem relaxation by removing
  2. … all preconditions
  3. … all preconditions <u>and</u> negative effects
  4. … negative effects only: Empty-Delete-List

# 1. Subgoal Independence Assumption

- The cost of solving a conjunction of subgoals is the sum of the costs of solving each subgoal independently
- Optimistic
  - Where subplans interact negatively
  - Example: one action in a subplan delete goal achieved by an action in another subplan
- Pessimistic (not admissible)
  - Redundant actions in subplans can be replaced by a single action in  merged plan

# 2. Problem Relaxation: Removing Preconditions

- Remove preconditions from action descriptions
  - All actions are applicable
  - Every literal in the goal is achievable in one step
- Number of steps to achieve the conjunction of literals in the goal is equal to the number of unsatisfied literals
- Alert
  - Some actions may achieve several literals
  - Some action may remove the effect of another action

# 3. Remove Preconditions & Negative Effects

- Considers only positive interactions among actions to achieve multiple subgoals
- The minimum number of actions required is the sum of the union of the actions' positive effects that satisfy the goal
- The problem is reduced to a set cover problem, which is NP-hard
  - Approximation by a greedy algorithm cannot guarantee an admissible heuristic

# 4. Removing Negative Effects (Only)

- Remove all negative effects of actions (no action may destroy the effects of another)
- Known as the Empty-Delete-List heuristic
- Requires running a simple planning algorithm
- Quick & effective
- Usable in progression or regression planning

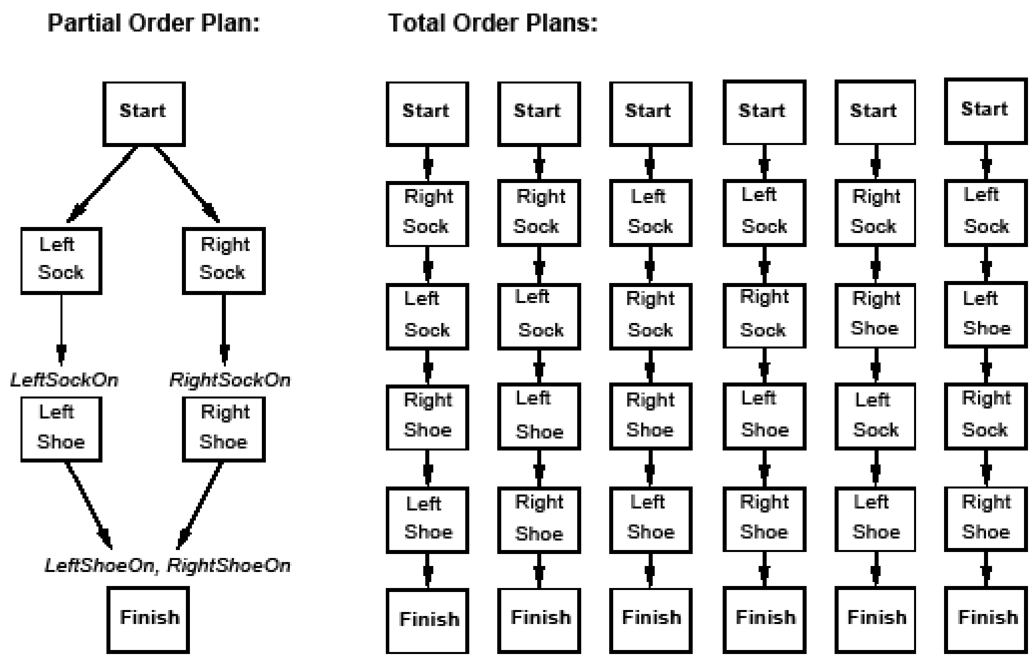
# Outline

- Background
  - Situation Calculus
  - Frame, qualification, & ramification problems
- Representation language
- **Planning Algorithms**
  - State-Space Search
  - **Partial-Order Planning (POP)**
  - Planning Graphs (GRAPHPLAN)
  - SAT Planners

# Partial Order Planning (POP)

- State-space search
  - Yields totally ordered plans (linear plans)
- POP
  - Works on subproblems independently, then combines subplans
  - Example
    - Goal(RightShoeOn $\wedge$ LeftShoeOn)
    - Init()
    - *Action*(RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
    - *Action*(RightSock, EFFECT: RightSockOn)
    - *Action*(LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
    - *Action*(LeftSock, EFFECT: LeftSockOn)

# POP Example & its linearization



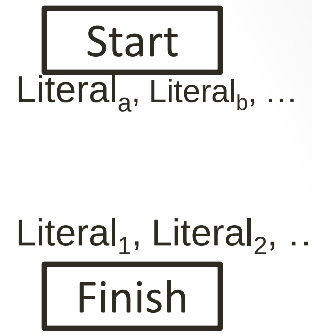
# Components of a Plan

1. A set of actions
2. A set of ordering constraints
   - A $\prec$ B reads "A before B" but not necessarily immediately before B
   - Alert: caution to cycles A $\prec$ B and B $\prec$ A
3. A set of causal links (protection intervals) between actions
   - A $\xrightarrow{p}$ B reads "A achieves *p* for B" and p must remain true from the time A is applied to the time B is applied
   - Example "RightSock $\xrightarrow{RightSockOn}$ RightShoe
4. A set of open preconditions
   - Planners work to reduce the set of open preconditions to the empty set w/o introducing contradictions

# Consistent Plan (POP)

- Consistent plan is a plan that has
  - No cycle in the ordering constraints
  - No conflicts with the causal links
- Solution
  - Is a consistent plan with no open preconditions
- To solve a conflict between a causal link $A \xrightarrow{p} B$ and an action C (that clobbers, threatens the causal link), we force C to occur outside the "protection interval" by adding
  - the constraint $C \prec A$ (demoting C) or
  - the constraint $B \prec C$ (promoting C)

# Setting up the PoP



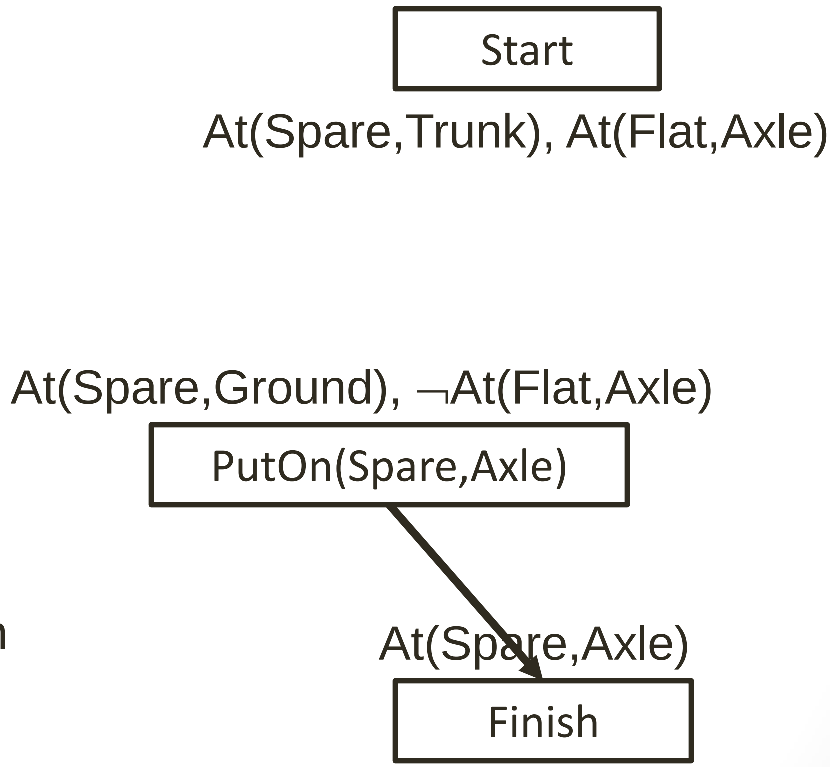

- Add dummy states
  - Start
    - Has no preconditions
    - Its effects are the literals of the initial state
  - Finish
    - Its preconditions are the literals of the goal state
    - Has no effects
- Initial Plan:
  - Actions: {Start, Finish}
  - Ordering constraints: {Start $\prec$ Finish}
  - Causal links: {}
  - Open Preconditions: {LeftShoeOn,RightShoeOn}

# POP as a Search Problem

- The successor function arbitrarily picks one open precondition *p* on an action B
- For every possible consistent action A that achieves *p*
  - It generates a successor plan adding the causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$
  - If A was not in the plan, it adds $Start \prec A$ and $A \prec Finish$
  - It resolves all conflicts between
    - the new causal link and all existing actions
    - between A and all existing causal links
  - Then it adds the successor states for combination of resolved conflicts
- It repeats until no open precondition exists

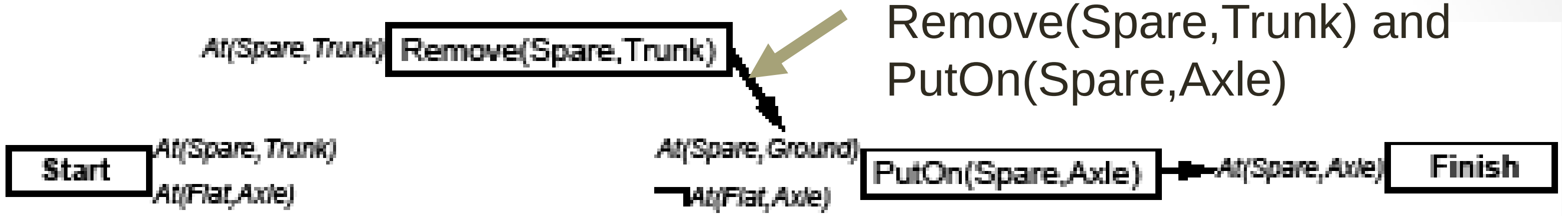
# Example of POP: Flat tire problem

- See problem description in Fig 11.7 page 391



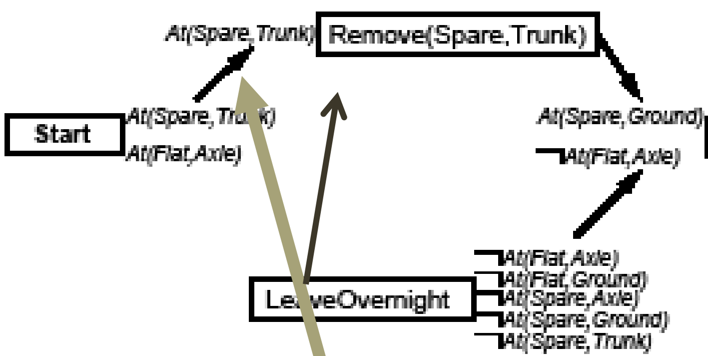

- Only one open precondition
- Only 1 applicable action

- Pick up At(Spare,Ground)
- Choose only applicable action Remove(Spare,Trunk)



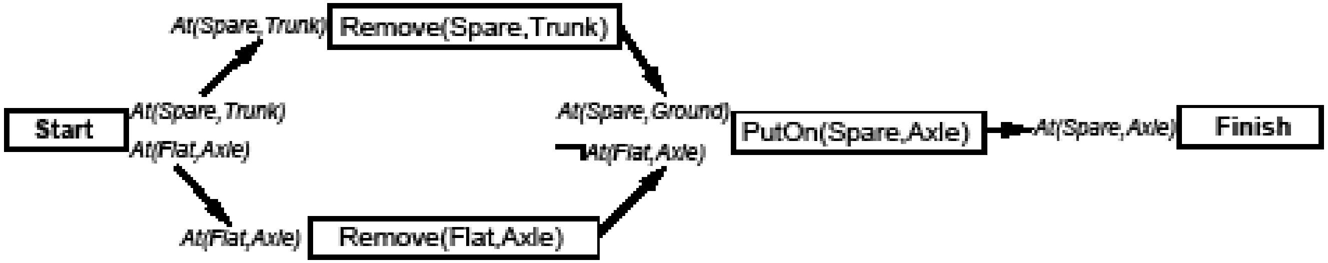

Add causal link between Remove(Spare,Trunk) and PutOn(Spare,Axle)

- Pick up ¬At(Flat,Axle)
- There are 2 applicable actions: LeaveOvernight and Remove(Flat,Axle)
- Choose LeaveOvernight




- LeaveOvernight has effect ¬At(Spare,Ground), which conflicts with the causal link
- We remove the conflict by forcing LeaveOvernight to occur before Remove(Spare,Trunk)

- Conflicts with effects of Remove(Spare,Trunk)
- The only way to resolve the conflict is to undo LeaveOvernightuse the action Remove(Flat,Axle)




- This time, we choose Remove(Flat,Axle)
- Pick up At(Spare,Trunk) and choose Start to achieve it
- Pick up At(Flat,Axle) and choose Start to achieve it.
- We now have a complete consistent partially ordered plan

# POP Algorithm (1)

- Backtrack when fails to resolve a threat or find an operator
- Causal links
  - Recognize when to abandon a doomed plan without wasting time expanding irrelevant part of the plan
  - allow early pruning of inconsistent combination of actions
- When actions include variables, we need to find appropriate substitutions
  - Typically we try to delay commitments to instantiating a variable until we have no other choice (least commitment)
- POP is sound, complete, and systematic (no repetition)

# POP Algorithm (2)

- Decomposes the problem (advantage)
- But does not represent states explicitly: it is hard to design heuristic to estimate distance from goal
  - Example: Number of open preconditions – those that match the effects of the start node.  Not perfect (same problems as before)
- A heuristic can be used to choose which plan to refine (which precondition to pick-up):
  - Choose the most-constrained precondition, the one satisfied by the least number of actions.  Like in CSPs!
  - When no action satisfies a precondition, backtrack!
  - When only one action satisfies a precondition, pick up the precondiction.

# Outline

- Background
  - Situation Calculus
  - Frame, qualification, & ramification problems
- Representation language
- **Planning Algorithms**
  - State-Space Search
  - **Partial-Order Planning (POP)**
  - **Planning Graphs (GRAPHPLAN)**
  - SAT Planners

# Planning Graph

- Is special data structure used for
  1. Deriving better heuristic estimates
  2. Extract a solution to the planning problem: GRAPHPLAN algorithm
- Is a 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
  - Starting at initial state
  - State level is a set of (propositional) literals
    - All those literals that could be true at that level
  - Action level is a set of (propositionalized) actions
    - All those actions whose preconditions appear in the state level (ignoring all negative interactions, etc.)
- Propositionalization may yield combinatorial explosion in the presence of a large number of objects

# Focus

- Building the Planning Graph
- Using it for Heuristic Estimation
- Using it for generating the plan

# Example of a Planning Graph (1)

Init(Have(Cake))

Goal(Have(Cake)∧Eaten(Cake))

Action(Eat(Cake))

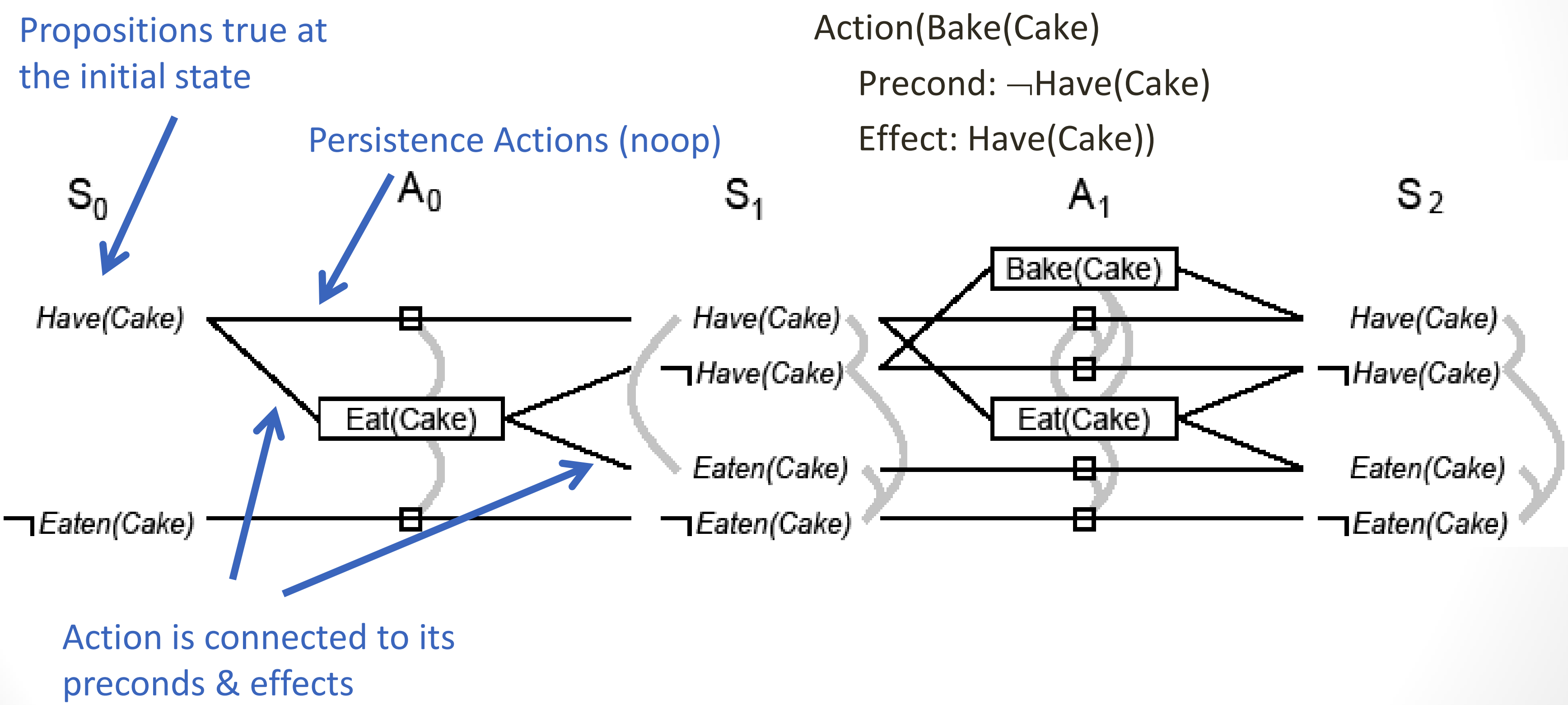
Precond: Have(Cake)

Effect: ¬Have(Cake)∧Eaten(Cake))

Action(Bake(Cake))

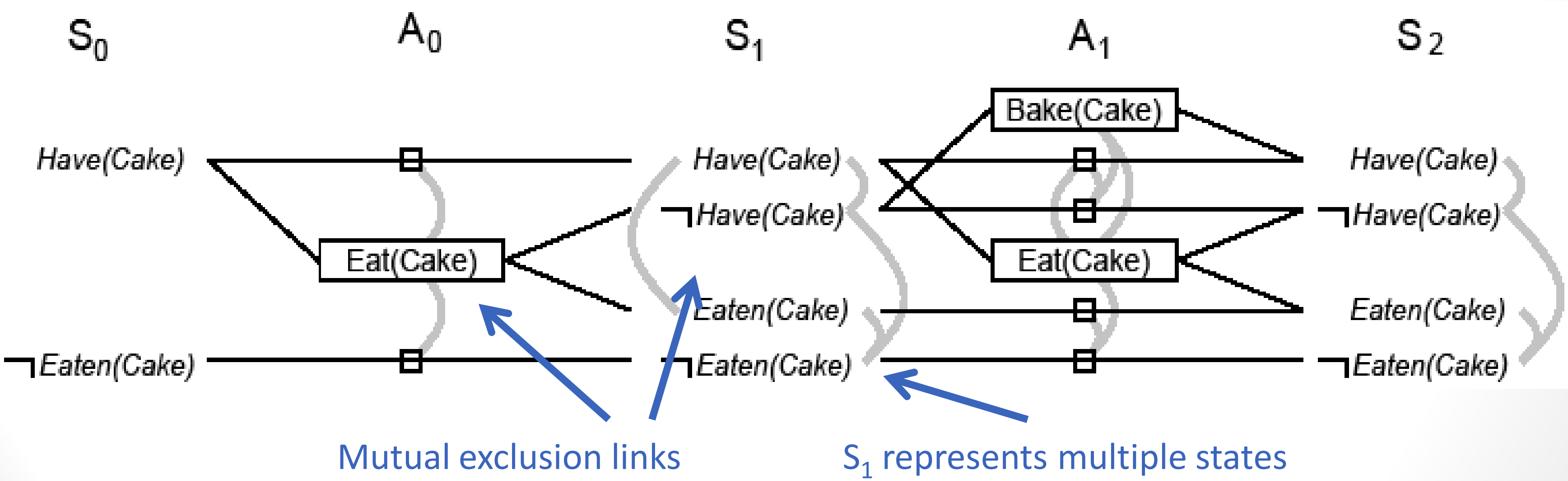
Precond: ¬Have(Cake)

Effect: Have(Cake))



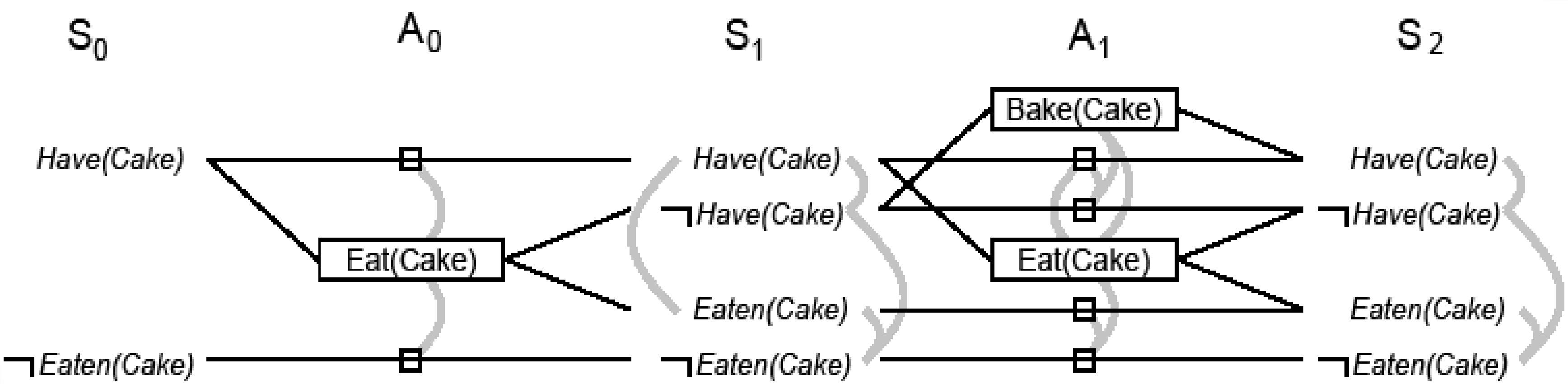
# Example of a Planning Graph (2)

- 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 process
- Add (binary) mutual exclusion (mutex) links between conflicting actions and between conflicting literals



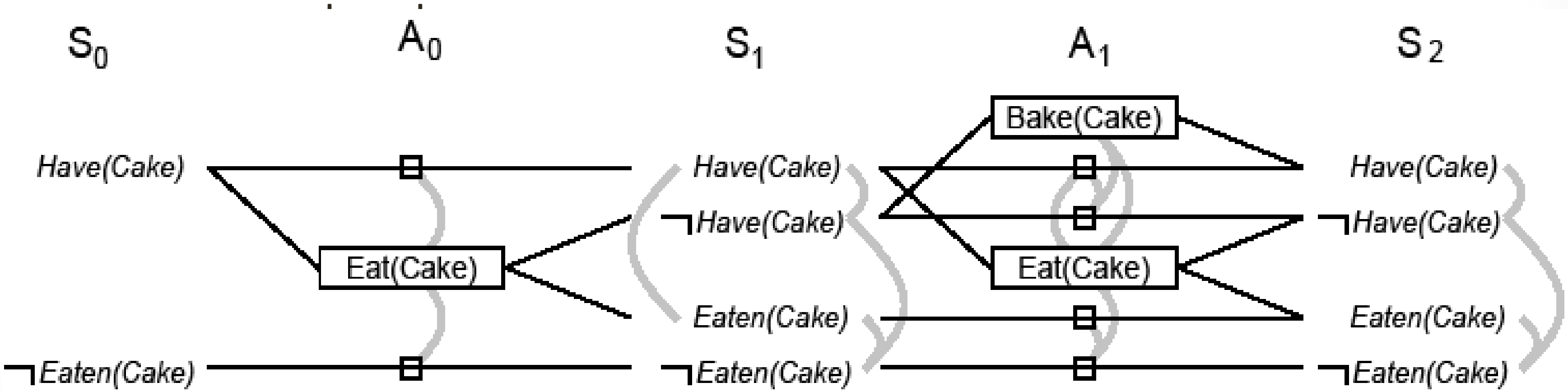
# Mutex Links between Actions

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



# 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)



# Focus

- Building the Planning Graph
- **Using it for Heuristic Estimation**
  - Planning graph as a relaxation of original problem
  - Easy to build (compute)
- Using it for generating the plan

# Planning Graph 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 $h(n)=\propto$
  - POP: Any plan with an unachievable open condition has cost $h(n)=\propto$
- The estimate 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

# Focus

- Building the Planning Graph
- Using it for Heuristic Estimation
- **Using it for generating the plan**
  - GraphPlan algorithm [Blum & Furst, 95]

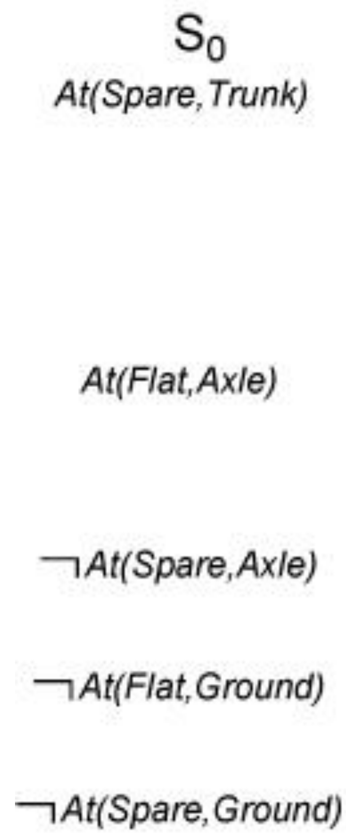
# GRAPHPLAN algorithm

```
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)
```

- Two main stages
  1. Extract solution
  2. Expand the graph

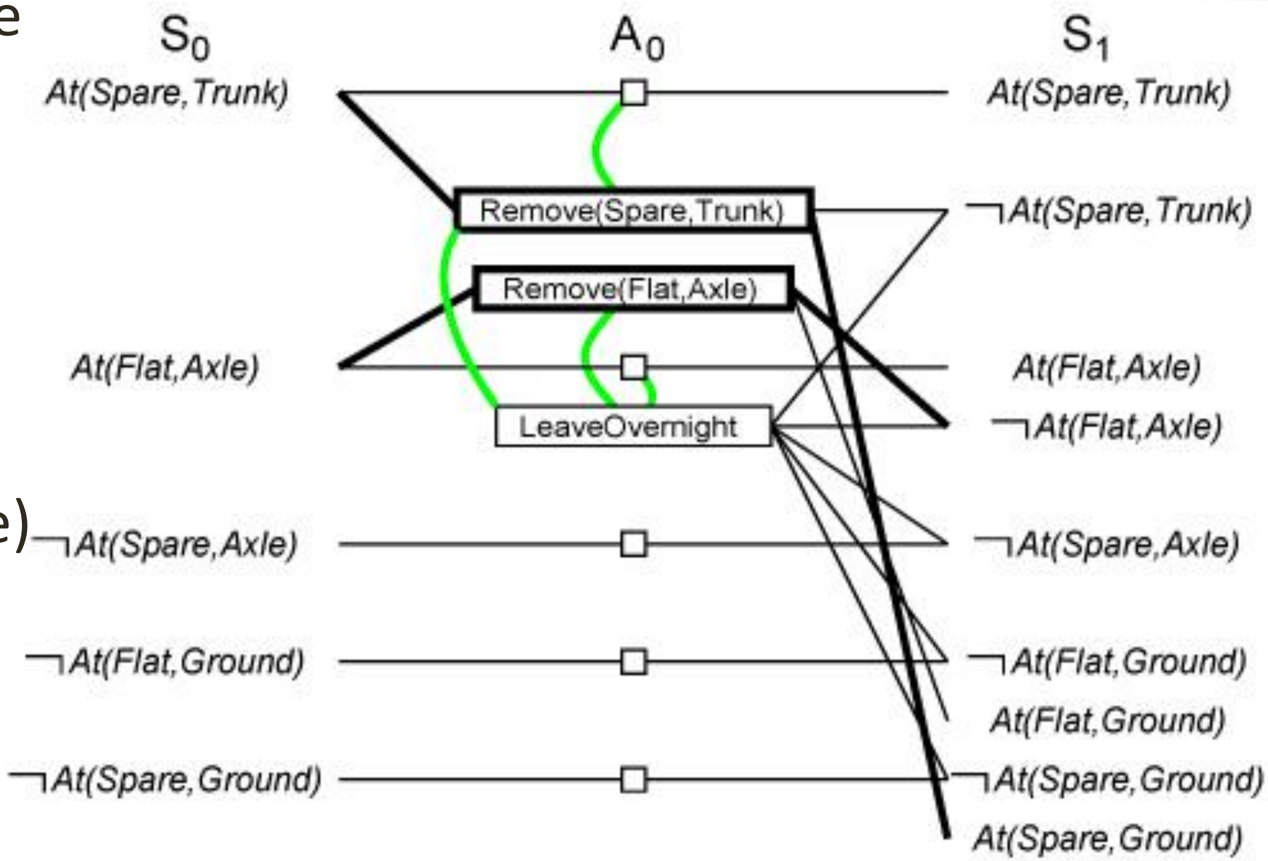
# Example of GRAPHPLAN Execution (1)

- At(Spare,Axle) is not in $S_0$
- No need to extract solution
- Expand the plan



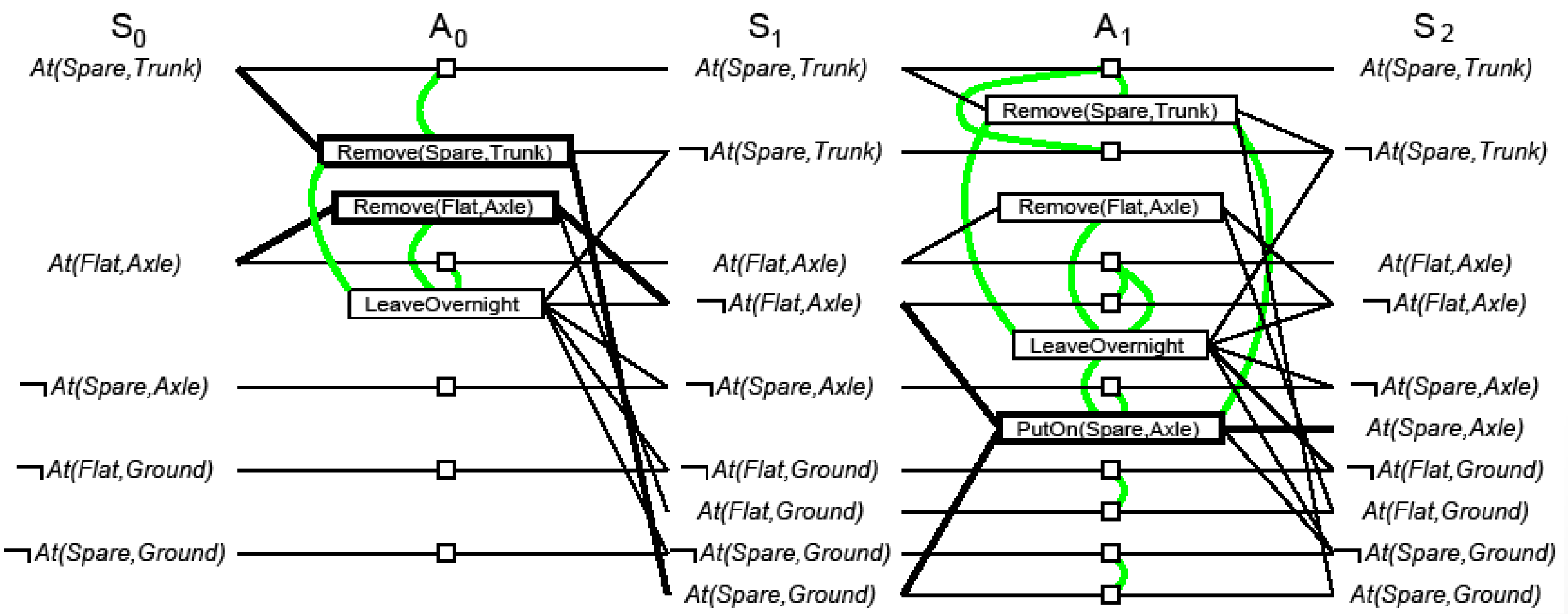
# Example of GRAPHPLAN Execution (2)

- 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



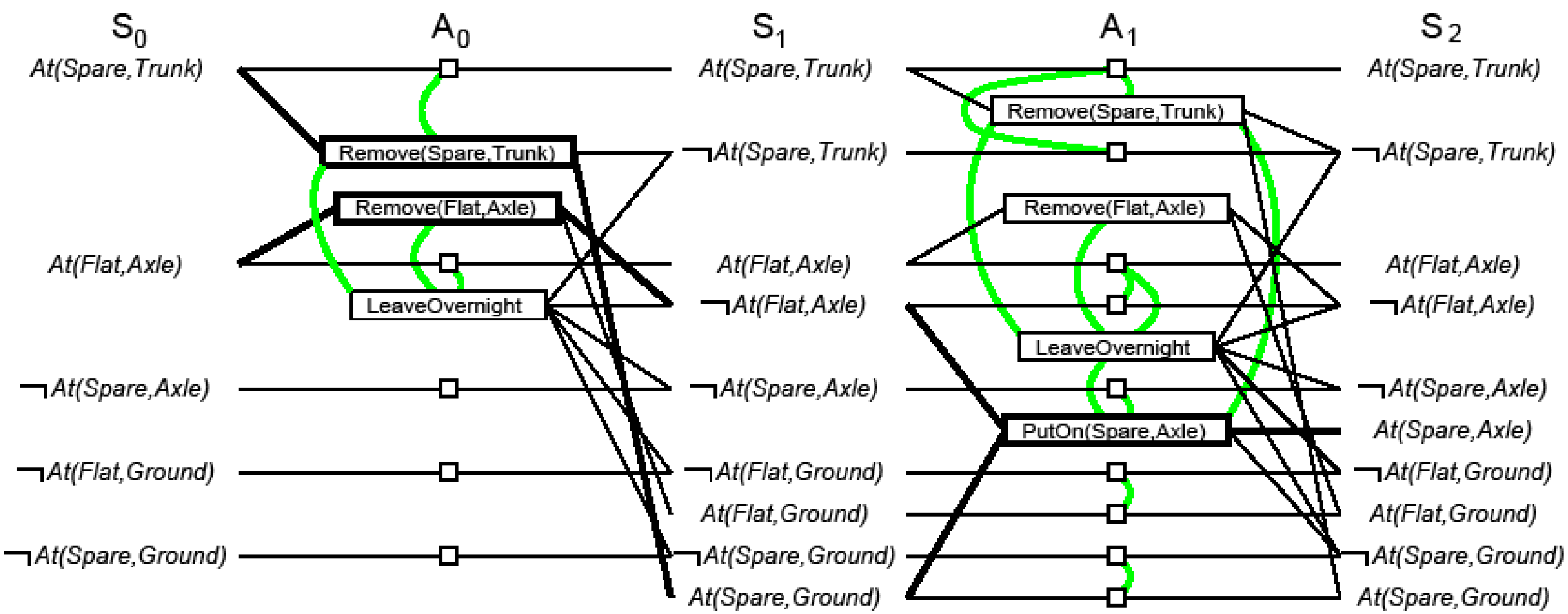
# Example of GRAPHPLAN Execution (3)

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



# Solution Extraction (Backward)

1. Solve a Boolean CSP: Variables are actions, domains are {0=out of plan, 1=in plan), constraints are mutex
2. Search problem from last level backward



# Backtrack Search for Solution Extraction

- Starting at the highest fact level
  - Each goal is put in a goal list for the current fact layer
  - Search iterates thru each fact in the goal list trying to find an action to support it which is not mutex with any other chosen action
  - When an action is chosen, its preconditions are added to the goal list of the lower level
  - When all facts in the goal list of the current level have a consistent assignment of actions, the search moves to the next level
- Search backtracks to the previous level when it fails to assign an action to each fact in the goal list at a given level
- Search succeeds when the first level is reached.

# 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)

# Outline

- Background
  - Situation Calculus
  - Frame, qualification, & ramification problems
- Representation language
- Planning Algorithms
  - State-Space Search
  - Partial-Order Planning (POP)
  - Planning Graphs (GRAPHPLAN)
  - SAT Planners