Introduction to AI

Lecture 11
Planning Graphs

Dr. Tamal Ghosh
Department of CSE
Adamas University

Planning Graph

- It is an algorithm for automated planning, developed by Avrim and Merrick in 1995.
- The Graph Plan's input is planning problem, expressed in STRIPS and produces a sequence of operations for reaching a goal state.

Planning Graph Algorithm

- Convert the planning problem structure into planning graph called as GRAPHPLAN, in the increment nature.
- It gives the relation between action and states, the precondition must be satisfy the action.
- The Planning graph is a layered graph, with alternate layers of propositions and actions.
  - Layer p0
  - Layer a1
  - Layer p1

Planning Graph Algorithm

- Propositional problem will look at, what the starting state, what the objects in the domains are, and it will produce all the possible actions, and works with those actions.
- We construct the planning graph from left to right,
  - we keep inserting actions and propositions, and
  - then actions and propositions
  - until we get the goal proposition appear on the proposition layer, and
  - they are not mutually exclusive.

Planning Graph Algorithm

- There are two states in the planning graph problem
  - Construct the planning graph
  - Search for solution
- If you cannot get solution, then extend the planning graph and search for solution, and keep doing that until you get the solution.

Planning Graph

- Planning Graph can give better heuristic estimates.
- Here we can extract a solution directly from the planning graph, using a specialized algorithm such as GRAPHPLAN
- A planning graph consists of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state.
- Each level contains a set of literals and a set of actions.
- The literals are true at that time step, depending on, the actions executed at preceding time steps.
- Actions *could* have their preconditions, that should be satisfied at that time step, depending on the literals actually hold.

Planning Graph

- Planning graphs are an efficient way to create a representation of a planning problem, that can be used to
  - Achieve better heuristic estimates
  - Directly construct plans
- Planning graphs only work for propositional problems.

Planning Graph

- It consists of a seq of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions that represent, what might be possible at that step in the plan
  - Records only a restricted subset of possible negative interactions among actions.

Planning Graph

- Each level consists of
  - Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.
  - Actions = all those actions have their preconditions, that satisfied at that time step, depending on which of the literals actually hold.

Example - The “have cake and eat cake too” problem.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \land Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\text{PRECOND: } Have(Cake)$$
$$\text{EFFECT: } \neg Have(Cake) \land Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\text{PRECOND: } \neg Have(Cake)$$
$$\text{EFFECT: } Have(Cake)$$

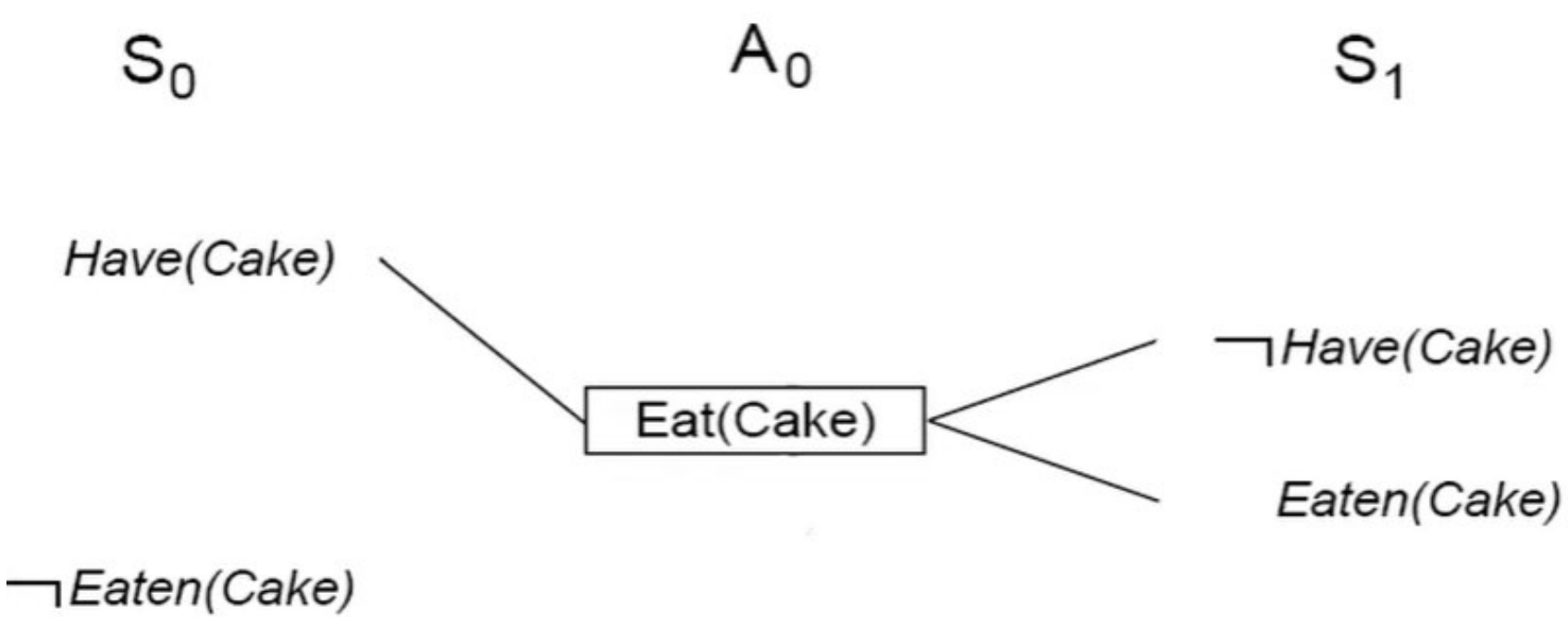
PG – example

S_0 A_0 S_1

Have(Cake)

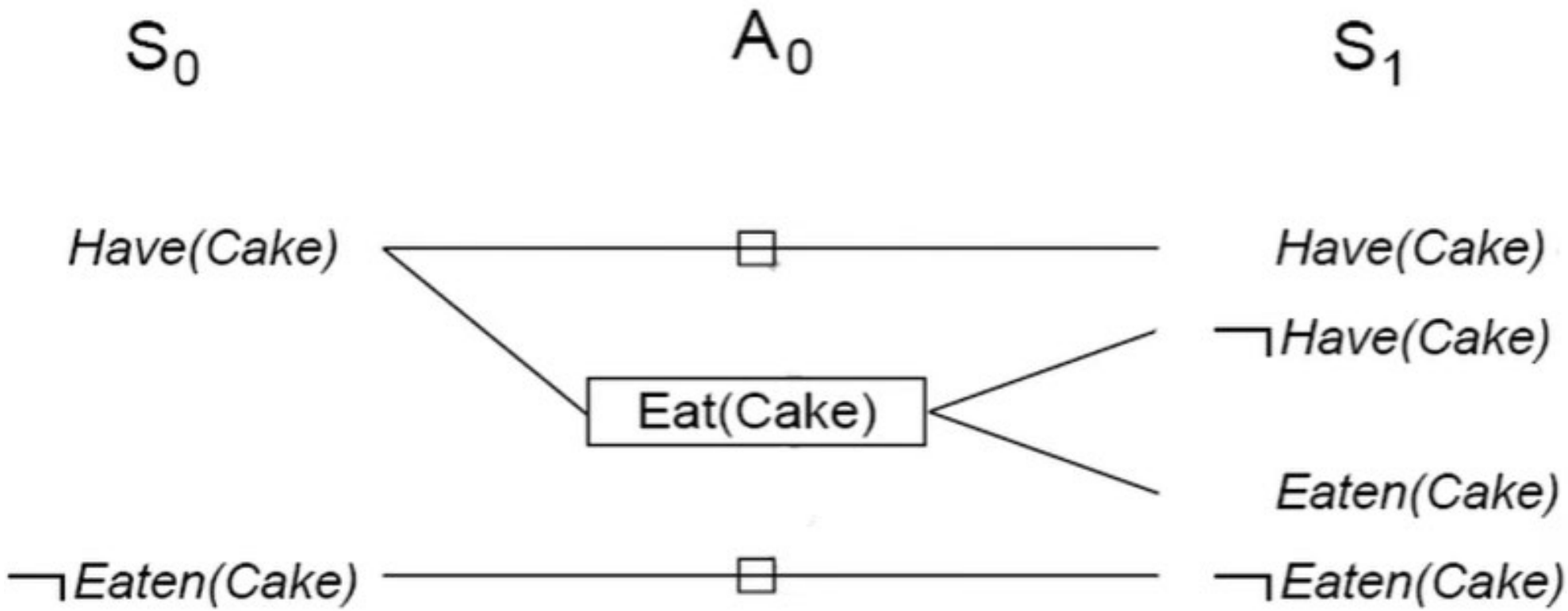
¬Eaten(Cake)

Create level 0 from initial problem state.

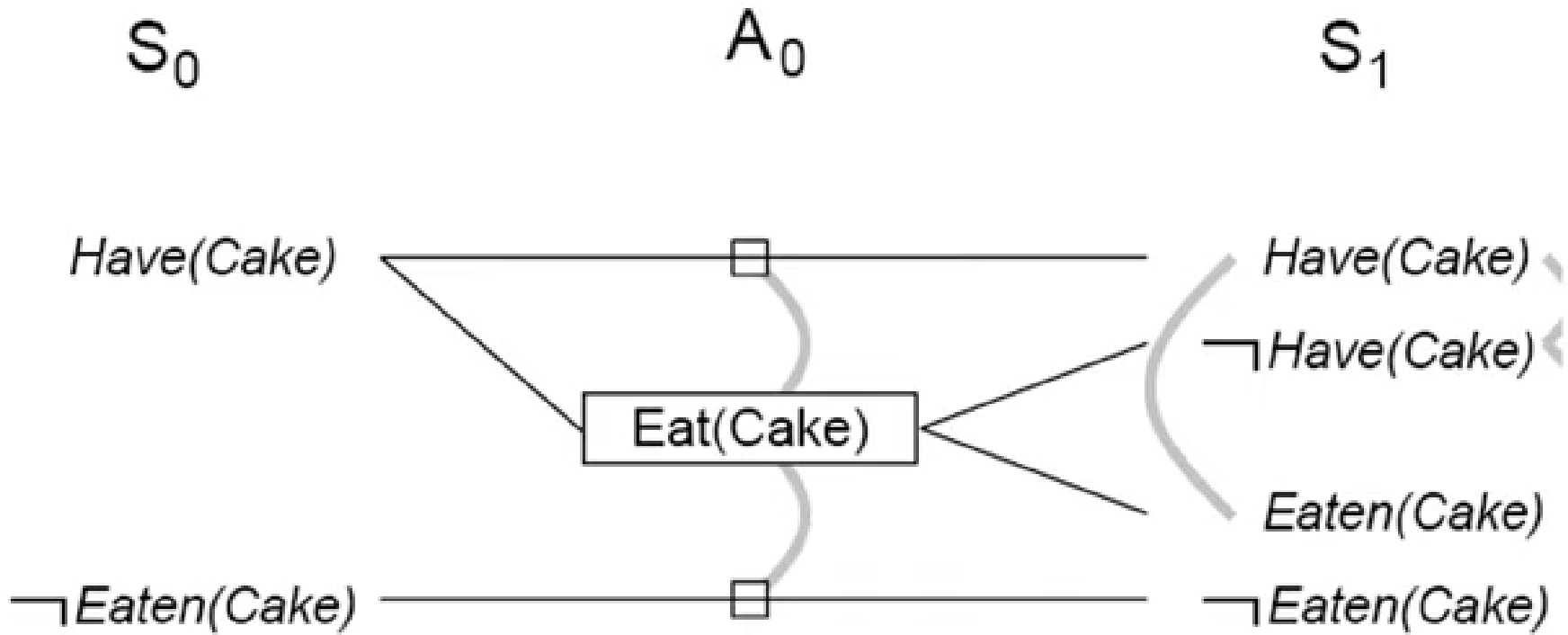


Add all applicable actions.

Add all effects to the next state.



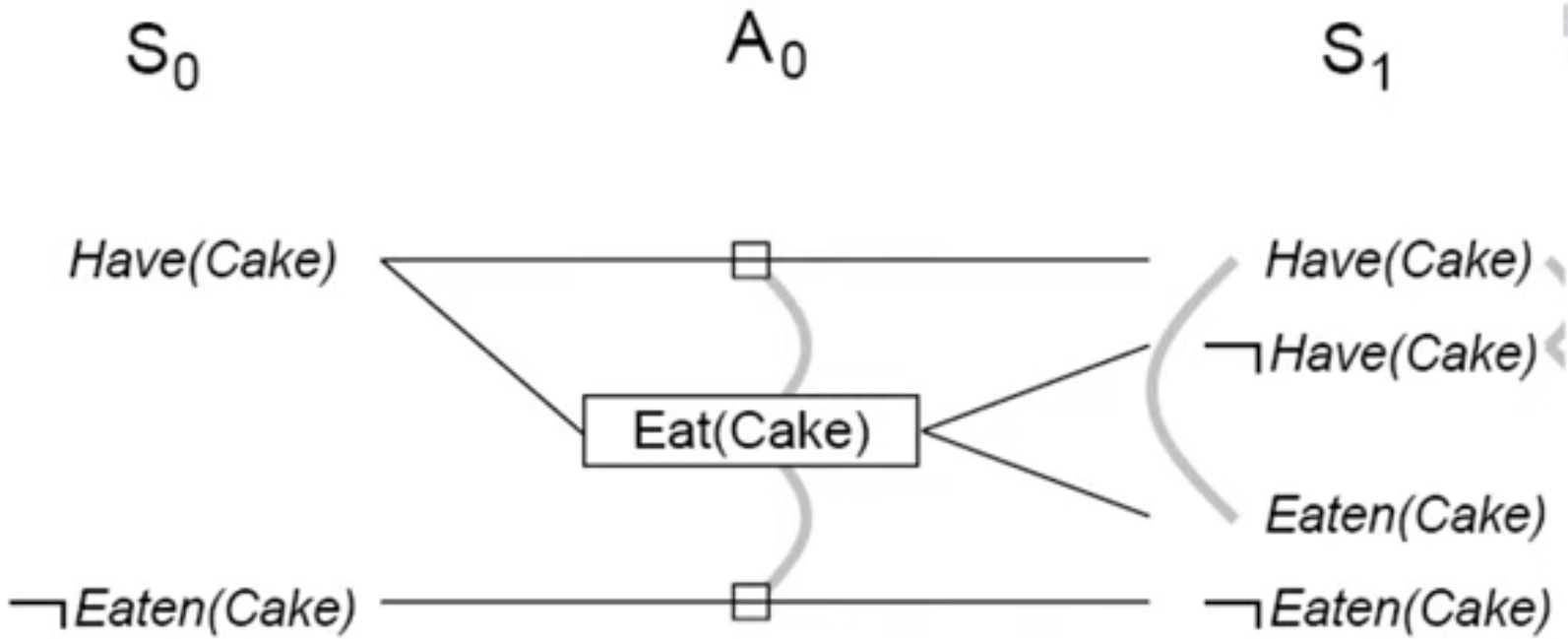
Add *persistence actions* (inaction = no-ops) to map all literals in state S_i to state S_{i+1}.



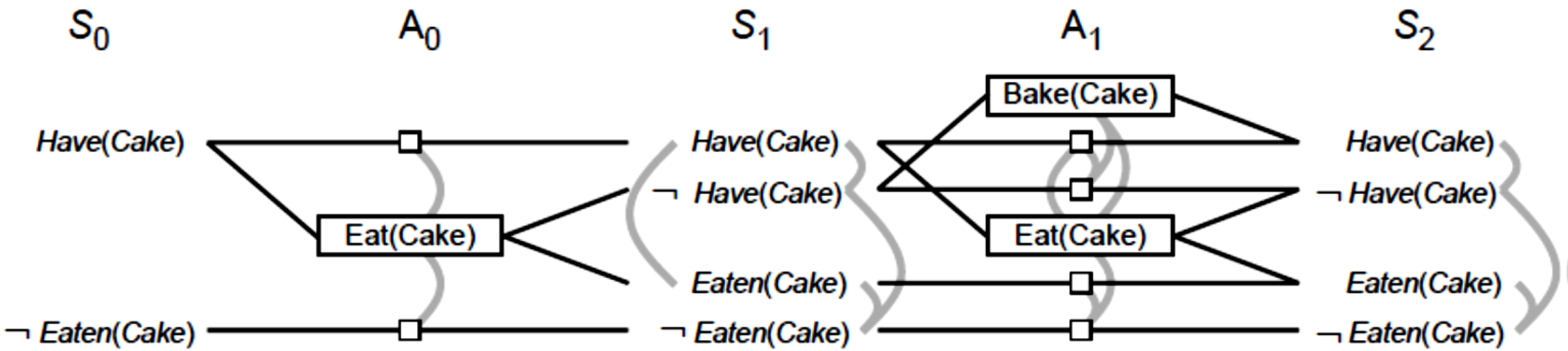
Identify *mutual exclusions* between actions and literals based on potential conflicts.

Mutual exclusion

- A mutex relation holds between two actions when:
  - Inconsistent effects: one action negates the effect of another.
  - Interference: one of the effects of one action, is the negation of a precondition of the other.
  - Competing needs: one of the preconditions of one action, is mutually exclusive with the precondition of the other.
- A mutex relation holds between two literals when:
  - one is the negation of the other
  - each possible action pair that could achieve the literals is mutex (inconsistent support).



- Level S1 contains all literals, that could result from, picking any subset of actions in A0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states, and the mutex links are the constraints, that define this set of states.



- Repeat process until graph levels off:
  - two consecutive levels are identical, or
  - contain the same amount of literals

PLANGRAPH termination

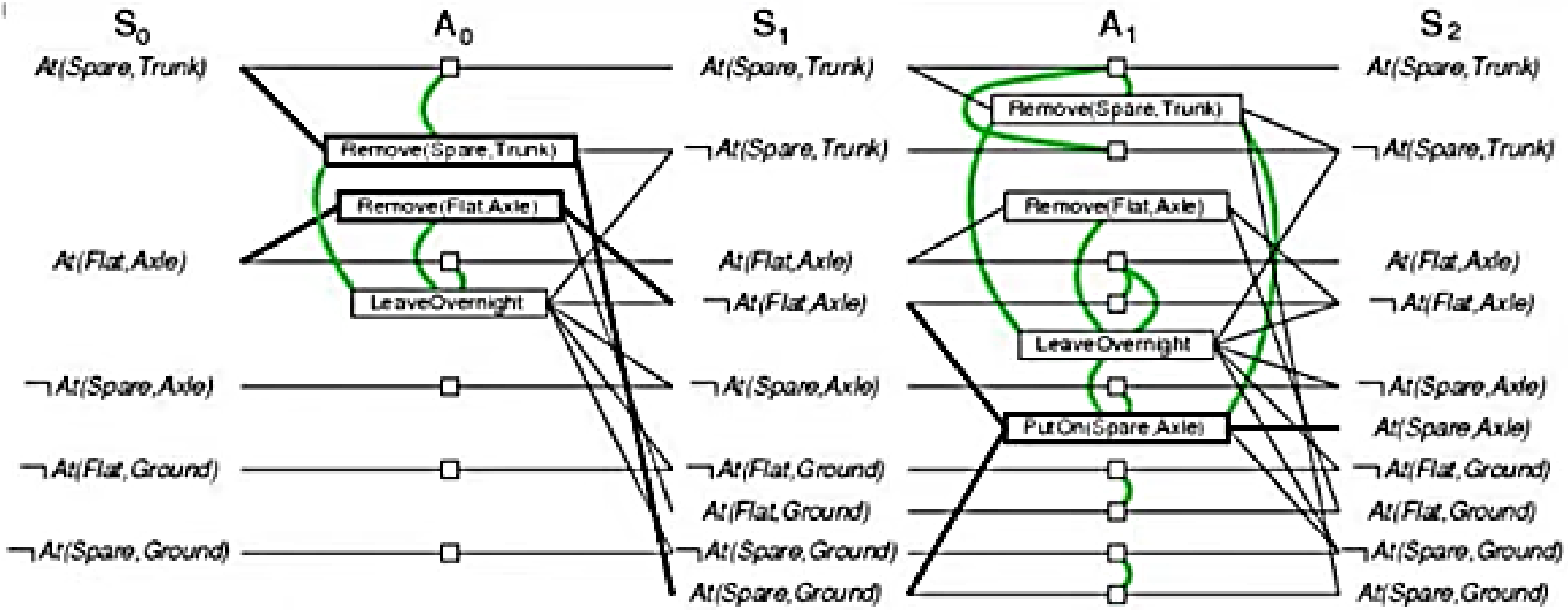
- Termination? YES
- PG are monotonically increasing or decreasing:
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotonically
- Because of these properties and a finite number of actions and literals, every PG will finally level off

The GRAPHPLAN Algorithm

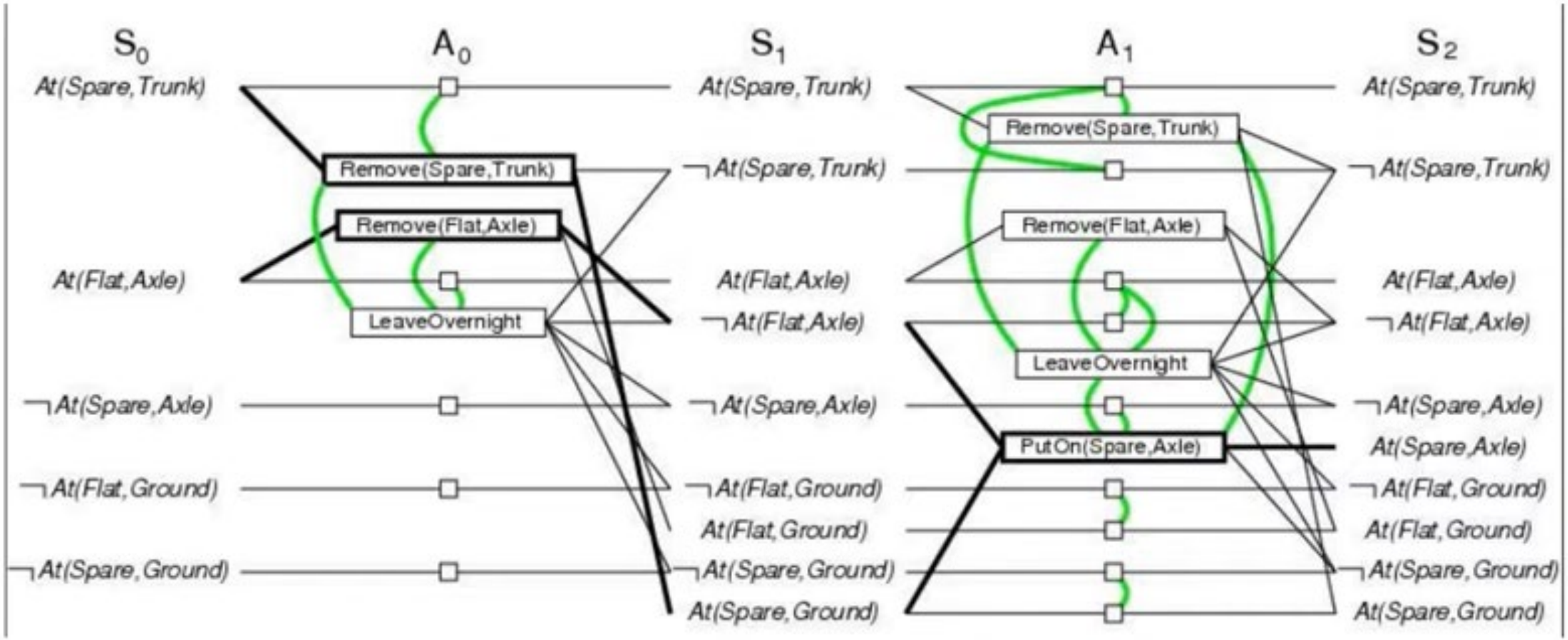
```
function GRAPHPLAN(problem) returns solution or failure

    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
            if solution ≠ failure then return solution
            else if NO-SOLUTION-POSSIBLE(graph) then return failure
        graph ← EXPAND-GRAPH(graph, problem)
```

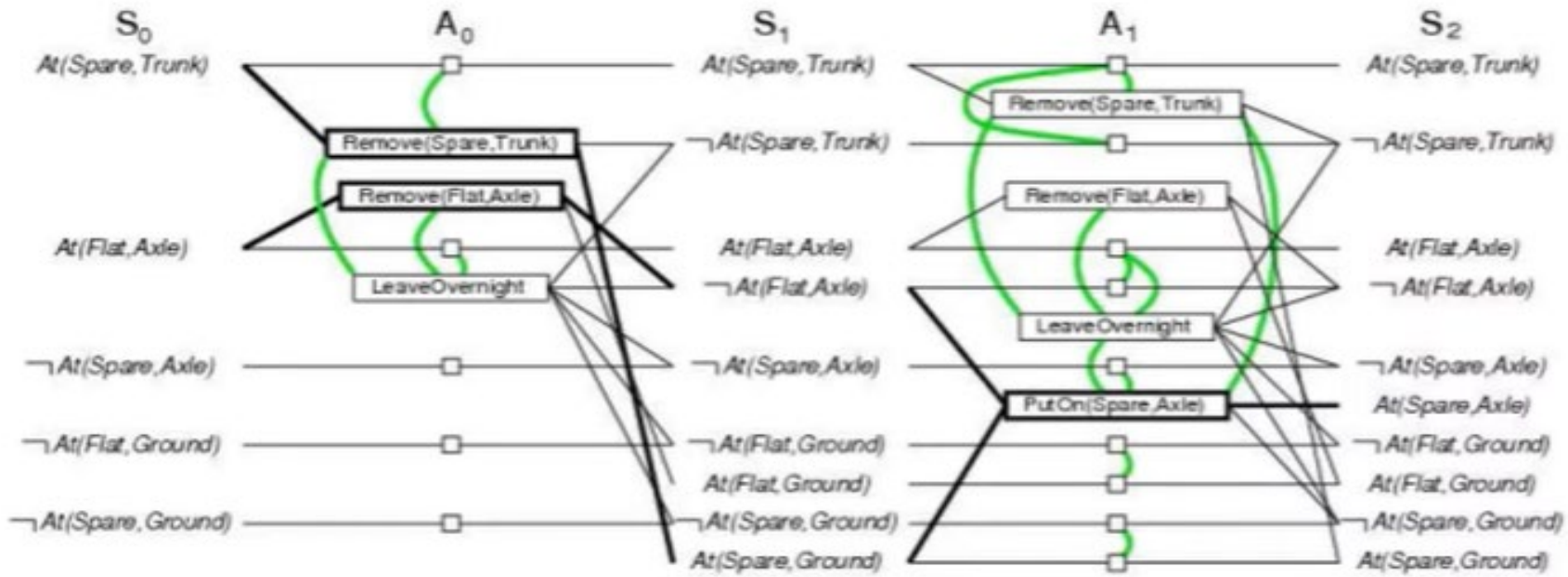
Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

GRAPHPLAN example – Change Flat Tire



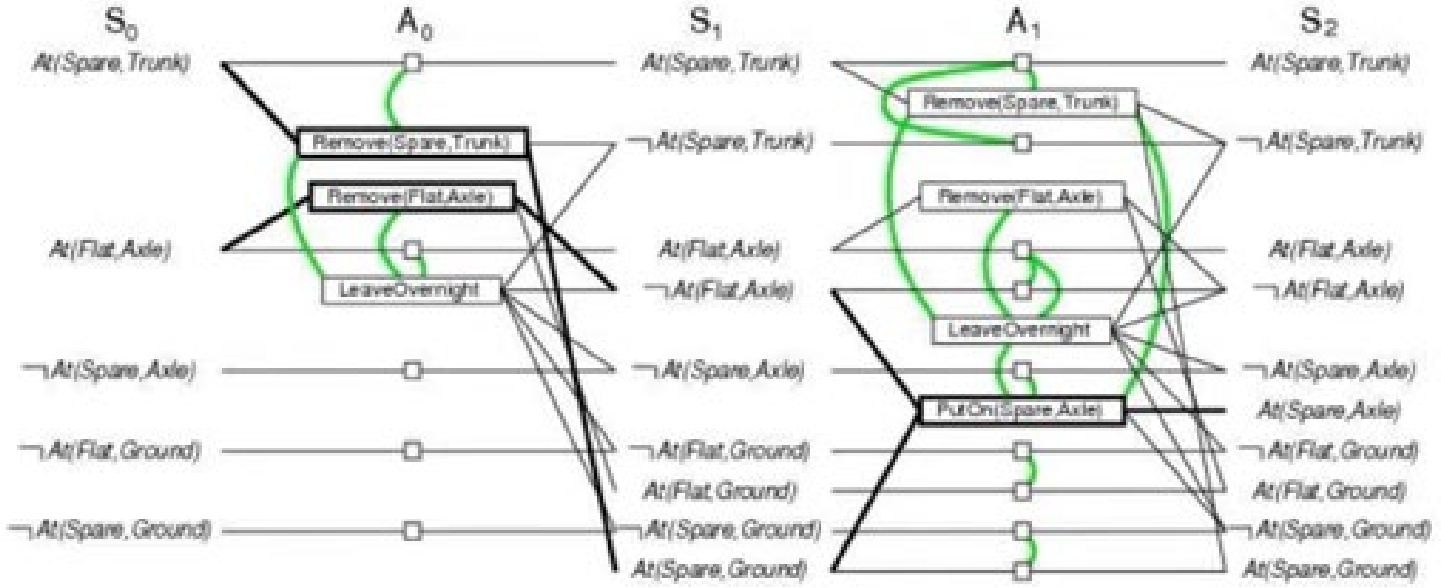
- Initially the plan consist of 5 literals from the initial state (S0).
- Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)
- Also add persistence actions and mutex relations.
- Add the effects at level S1
- Repeat until goal is in level Si


S0
A0
S1
A1
S2
At(Spare,Trunk)
Remove(Spare,Trunk)
Remove(Flat,Axle)
At(Flat,Axle)
LeaveOvernight
¬At(Spare,Axle)
¬At(Flat,Ground)
¬At(Spare,Ground)
At(Spare,Trunk)
¬At(Spare,Trunk)
At(Flat,Axle)
¬At(Flat,Axle)
¬At(Spare,Axle)
¬At(Flat,Ground)
At(Flat,Ground)
¬At(Spare,Ground)
At(Spare,Ground)
At(Spare,Trunk)
Remove(Spare,Trunk)
Remove(Flat,Axle)
LeaveOvernight
PutOn(Spare,Axle)
At(Spare,Trunk)
¬At(Spare,Trunk)
At(Flat,Axle)
¬At(Flat,Axle)
¬At(Spare,Axle)
At(Spare,Axle)
¬At(Flat,Ground)
At(Flat,Ground)
¬At(Spare,Ground)
At(Spare,Ground)



EXPAND-GRAPH also looks for mutex relations

- Inconsistent effects
  - E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare,Ground) and **not** At(Spare, Ground)
- Interference
  - E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and **not** At(Flat,Axle) as EFFECT
- Competing needs
  - E.g. PutOn(Spare,Axle) and Remove(Flat, Axle) due to At(Flat.Axle) and **not** At(Flat, Axle)
- Inconsistent support
  - E.g. in S2, At(Spare,Axle) and At(Flat,Axle)



In S2, the goal literals exist and are not mutex with any other

- Solution might exist and EXTRACT-SOLUTION will try to find it

EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:

- Initial state = last level of PG and goal 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.

Planning in Real World

- Real-world domains are complex and don't satisfy the assumptions of STRIPS or partial-order planning methods:

  In complex real-world projects, it is common to use scheduling tools from Operations Research such as PERT charts or the critical path method. These tools essentially take a hand constructed complete partial-order plan and generate an optimal schedule for it.

  For most practical applications, however, there will be many related problems to solve, so it is worth the effort to describe the domain and then have the plans automatically generated. Especially important during the execution of plans. If a step of a plan fails, it is often necessary to replan quickly to get the project back on track. PERT charts do not contain the causal links and other information needed to see how to fix a plan, and human replanning is often too slow!

Planning in Real World

- Classical planners assume
  - √ Fully observable, static and deterministic domains
  - √ Correct and complete action descriptions
  - √ … allowing a "plan-first-then-act" planning approach
- … but in the real world
  - √ The world is dynamic, and *time* cannot be ignored
  - √ Information on the world is incomplete and incorrect
  - √ … the agent must be prepared for unexpected events
- Plus - scaling up to real-world problem size!

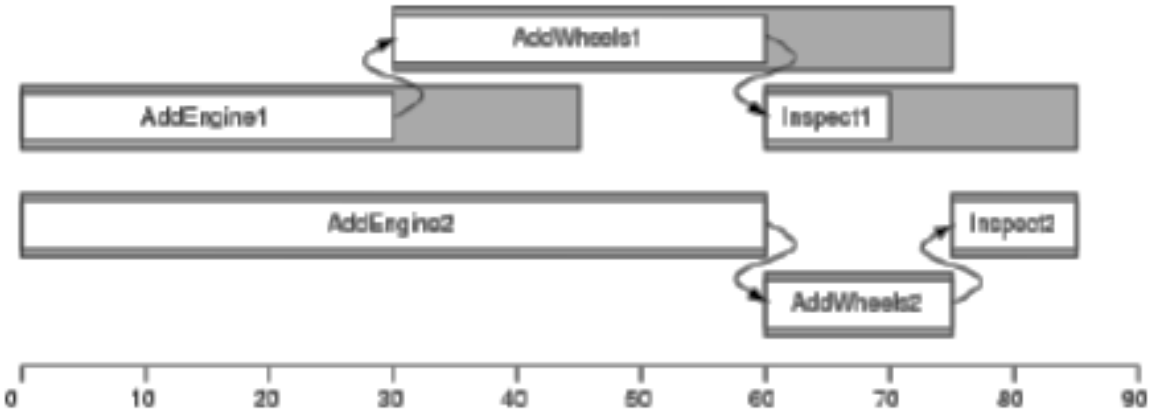
STRIPS insufficiency: Time, Schedules, Resources

- The PDDL language allows events (actions) and ordering of events, but not time *duration*
- In real-life planning, we must take duration, delays, etc. into account (not just ordering)
- *Job shop scheduling*:
  - The problem is to complete a set of jobs
  - Each job consists of a set of actions, with given duration and resource requirements
  - Determine a schedule that minimizes total time *(makespan)* needed while respecting resource constraints
- Must extend representation language to express duration and resource constraints

Scheduling: No resource constraints

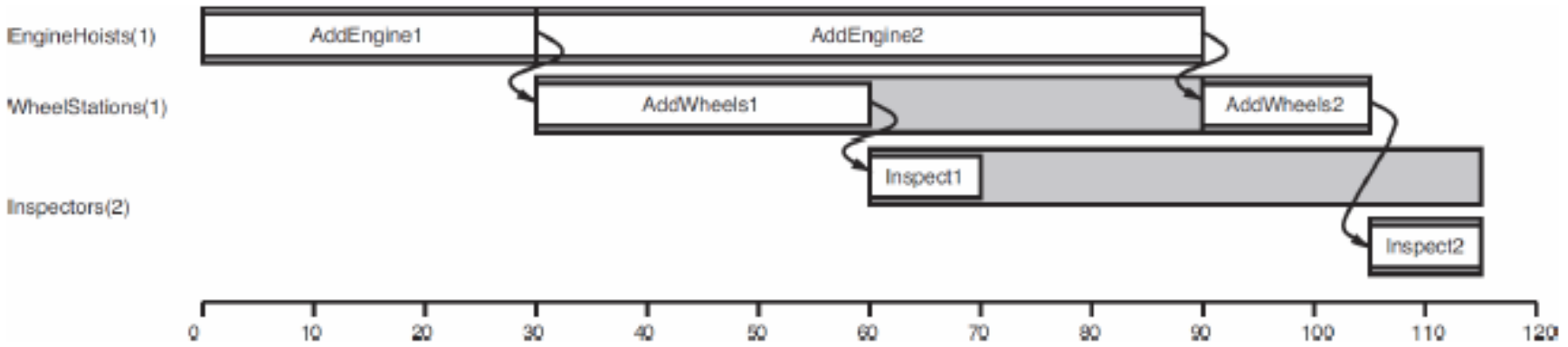
- Partial order plan produced by e.g. POP
- To create a *schedule*, we must place actions on a timeline
- Can use *critical path* method (CPM): the longest path, no slack – determines total duration
- Shortest duration schedule, given partial-order plan:

85 minutes



Scheduling: With resource constraints

- Actions typically require resources
  - √ *Consumable* resources – e.g. *LugNuts*
  - √ *Reusable* resources – e.g. *EngineHoists*
- Resource constraints make scheduling more complex because of interaction between actions
- AI and OR (Operations Research) methods can be used to solve scheduling problems with resources
- Shortest duration gone up from 85 to 115 minutes

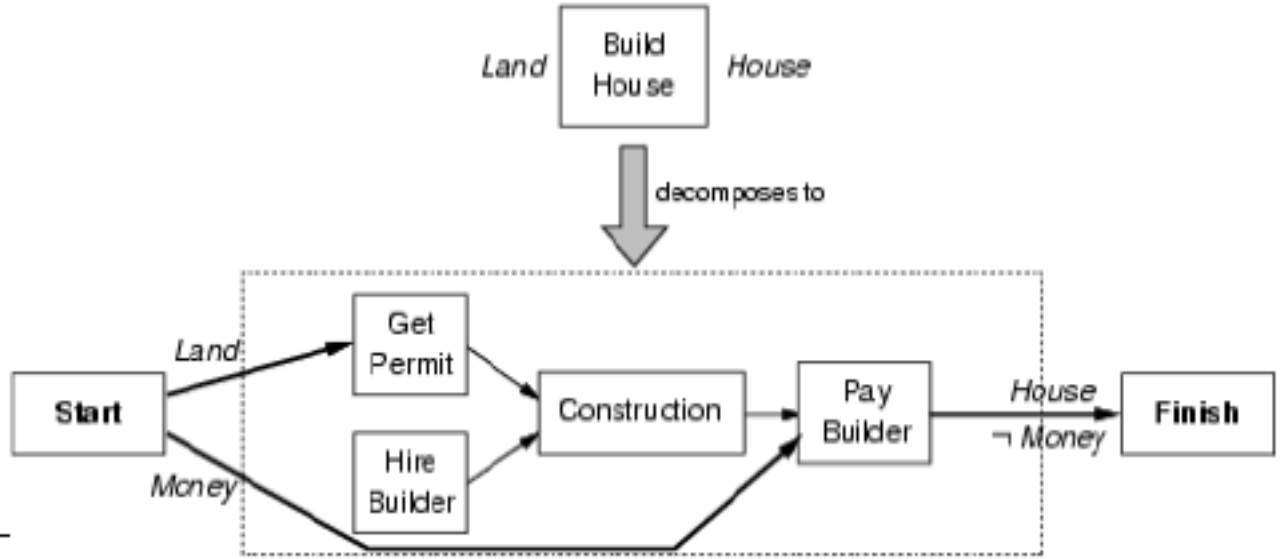


Planning and scheduling

- The approach shown here is common in real-world AI applications for manufacturing scheduling, airline scheduling, etc. :
  - √ First generate partial order plan without timing information (*planning*)
  - √ Then use separate algorithm to find optimal (or satisfactory) time behavior (*scheduling*)
- In some cases it may be better to *interleave* planning and scheduling, e.g. to consider temporal constraints already at the planning stage

Reduce complexity by decomposition

- Often possible to reduce problem complexity by decompose to subproblems, solve independently, and assemble solution
- HTN – Hierarchical Task Networks
  - √ Planner keeps library of subplans
  - √ Extend planning algorithm to use subplans
  - √ Can reduce time&space requirements considerably
- Most real-world planners use HTN variants



Planning in nondeterministic domains

- Nondeterministic worlds
  - *Bounded* nondeterminism: Effects can be enumerated, but agent cannot know in advance which one will occur
  - *Unbounded* nondeteminism: The set of possible effects is unbounded or too large to enumerate
- Planning for bounded nondeterminism
  - Sensorless planning
  - Contingent planning
- Planning for unbounded nondeterminism
  - Online replanning
  - Continuous planning

Sensorless planning

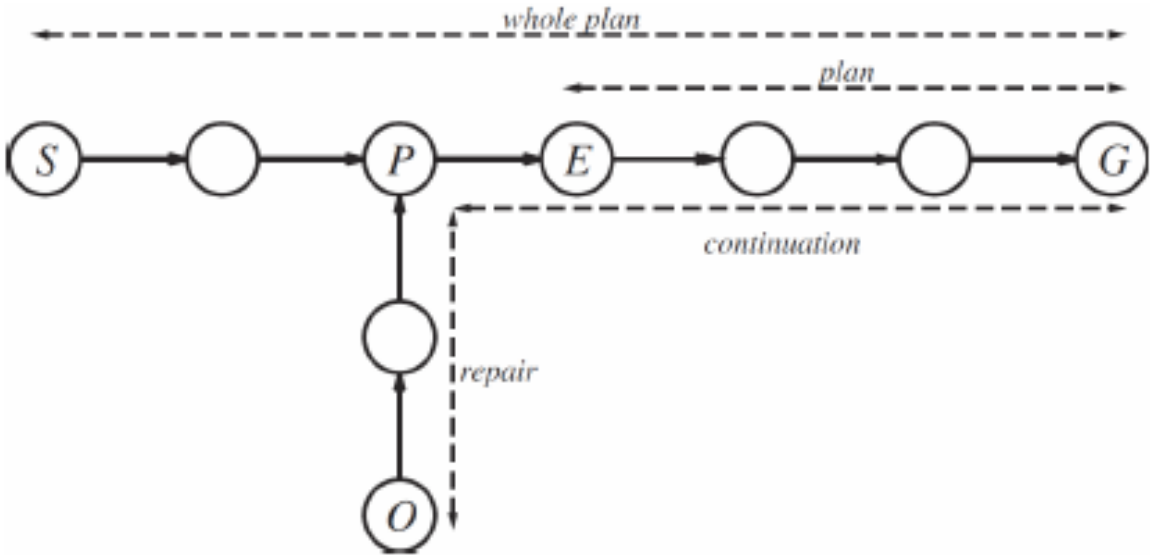
- Agent has no sensors to tell which state it is in, therefore each action might lead to one of several possible outcomes
- Must reason about *sets* of states (belief states), and make sure it arrives in a goal state regardless of where it comes from and results of actions
- Nondeterminism of the environment does not matter – the agent cannot detect the difference anyway
- The required reasoning is often not feasible, and sensorless planning is therefore often not applicable

Contingent planning

- Constructs conditional plans with branches for each (enumerable) possible situation
- Decides which action to choose based on special sensing actions that become parts of the plan
- Can also tackle partially observable domains by including reasoning about belief states (as in sensorless planning)
- Planning algorithms have been extended to produce conditional branching plans

Online replanning

- Monitors situation as plan unfold, detects when things go wrong
- Performs replanning to find new ways to reach goals, if possible by repairing current plan



- Agent proceeds from S, and next expects E following original *whole-plan*
- Detects that it's actually in O
- Creates a *repair* plan that takes it from O to a state P in original plan
- New plan to reach G becomes *repair + continuation*

Contingent Planning vs. Replanning

- Contingent planning
  - √ All actions in the real world have additional outcomes
  - √ Number of possible outcomes grows exponentially with plan size, most of them are highly improbable
  - √ Only one outcome will actually occur
- Replanning
  - √ Basically assumes that no failure occurs
  - √ Tries to fix problems as they occur
  - √ May produce fragile plans, hard to fix if things go wrong

Continuous Spectrum of Planners

- Contingent planning and replanning are extremes of a spectrum, where intermediate solutions exist
  - √ Disjunctive outcomes for actions where more than one outcome is likely
  - √ Agent can insert sensing action to detect what happened and construct corresponding conditional plan
  - √ Other contingencies dealt with by replanning
- More generally, agents in complex domains and with incomplete/incorrect information should
  - √ Assess likelihood and costs of various outcomes
  - √ Construct plan that maximizes probability of success and minimizes cost
  - √ Ignore contingencies that are unlikely or easy to deal with

Continuous Planning

- The planner persists over time – never stops, and interleaves planning, sensing and execution
- The continuously planning agent must
  - √ Execute steps of current plan (even if not complete)
  - √ Refine plan if not applicable or in conflict
  - √ Modify plan in light of new information
  - √ Formulate new goals when required
- Planners, e.g. partial-order planning (POP) can be extended to provide required functionality

Multi-Agent Planning

- Single-agent planning works against ”nature”, but in many cases the environment includes other agents with their own goals
- Multi-agent environments can be
  - *Cooperative*: Agents work together to achieve some common goal
  - *Competitive*: Agents have conflicting goals
- Multi-agent architectures and applications, incl. planning, represent very active AI research area

Coordination of multi-agent planning

- Cooperative planning can produce *joint plans*
  - For each agent, the joint plan tells what to do
  - If each follows its plan, overall goal will be achieved
- Problems arise if several joint plans are possible
  - Each agent must know which plan to follow
  - Requires some form of *coordination*
- Coordination can be by
  - Convention or *social law*
  - Inter-agent *communication*

In a nutshell

- Classical planning systems assume deterministic and static domains, and complete and correct information. Many domains violate this assumption
- *Scheduling* is planning with time and resource constraints and are solved by special methods
- Large planning problems can be made tractable by *hierarchic decomposition* (hierarchic task networks)
- Nondeterministic domains can be *bounded* (enumerable outcomes) or *unbounded* (any outcome is possible)

- Planning in bounded determinism includes *sensorless planning* (make sure plan succeeds) or *contingent planning* (select one of multiple pre-made plans bases on sensing)
- Planning in unbounded nondeterminism includes *online replanning* (repair plan if failure) or *continuous planning* (ongoing adaptation of plan)
- *Multi-agent planning* applies to domains where there are other *cooperative* or *competitive* agents