Classical Planning

Hypothesis

Hypothesis:

- Environment:
  - Fully observable,
  - deterministic,
  - static
- Single agents
- Factored representation: a collection of variables is employed to represent the state space.

Planning Domain Definition Language - PDDL

- PDDL aims to state a formalism when describing planning domains.
- PDDL facilitates to describe actions.
- A state is defined by a conjunction of fluents: $At(drone,base) \wedge At(cargo,base)$
- Fluents are basic and functionless atoms and those not mentioned are false.

  At(x,y) is not a fluent as well as At(Land(drone),base)

Planning Domain Definition Language - PDDL

- Actions are described by ACTIONS(s) and RESULT(s, a) functions.
- The result of an action describes what changes.
- Example of action schema:

  Action(Fly(p, from, to),
  PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  EFFECT: ¬At(p, from) ∧ At(p, to))

- Precondition and effect are conjunctions of literals, which can be positive or negated atomic sentences.

Planning Domain Definition Language - PDDL

- Actions are described by ACTIONS(s) and RESULT(s, a) functions.
- The result of an action describes what changes.
- Example of action schema:

  Action(Fly(p, from, to),
  PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  EFFECT: ¬At(p, from) ∧ At(p, to))

- Precondition and effect are conjunctions of literals, which can be positive or negated atomic sentences.

Planning Domain Definition Language - PDDL

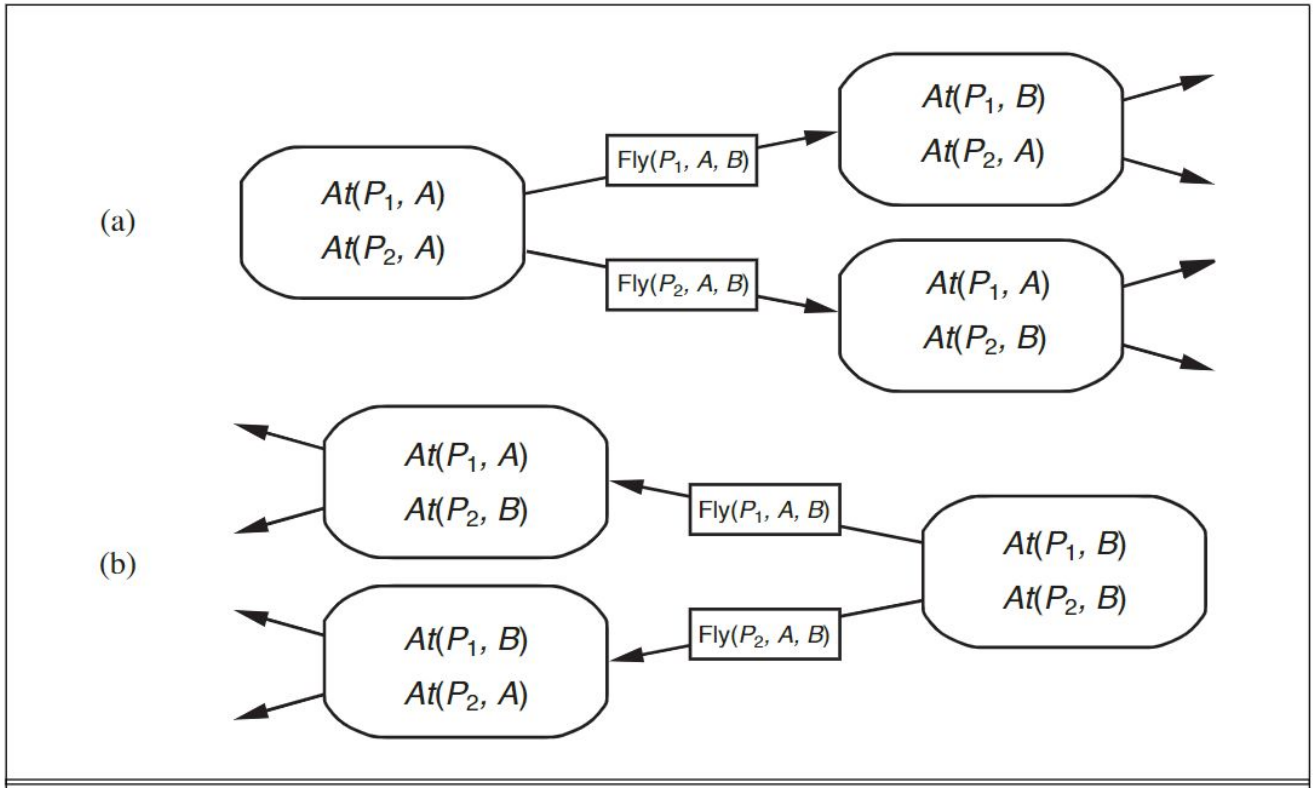
- The result of executing action behaviours as follows:
  - RESULT(s, a)=(s − DEL(a)) ∪ ADD(a)
  - DEL removes negative fluents from the effects of action
  - ADD adds positive fluents in the effects
- A planning domain is defined by a set of action schemas.
- The problem is defined in such domain by adding the initial and goal states.

Planning Domain Definition Language - PDDL

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Algorithm for Planning



Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Algorithm for Planning

Forward state-space:

- Inefficient: it will explore irrelevant actions within large state spaces in the planning problems context.

Backward relevant state-space:

- The actions relevant to the current state or the goal are evaluated.

Heuristic for Planning

- A relaxed version of the problem can be used to define an admissible heuristic function.
  - An admissible heuristic function does not return a value higher than the lowest value from the current state to the goal state.
- The heuristic function applied over the relaxed problem will become the heuristic function in the original problem.
- How to relax the problem:
  - First, dropping all preconditions from actions and all effects, except those that are literals in the goal. Next, counting the minimum number of actions required such that the union of those actions' effects satisfies the goal.
  - State abstraction: many-to-one mapping from states
  - Split the problem and solve each piece independently. Next, combining the solutions for each subgoal.

Planning Graphs

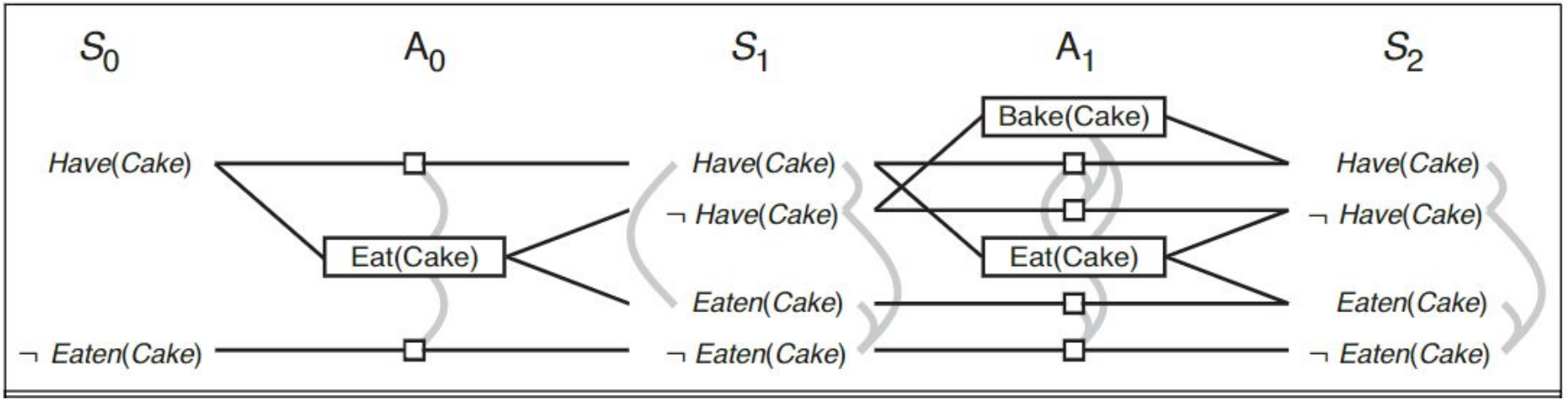
Planning graph: helps to find better heuristic estimates.

Planning graph is a directed graph organized into level, where the state S_i at level i has all the literals that hold at time i.

These literals are given by the previous actions executed.

Planning Graphs

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



Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

Planning Graph

In the planning graph generation:

- It is not necessary selecting actions, avoiding a combinatorial search.
- It is necessary to keep record of the not allowed choices through the mutex links.

Mutex conditions for two actions:

- Inconsistent effects: one action negates an effect of the other.
- 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 a precondition of the other.

Planning Graph

Mutex conditions between two literal:

- Inconsistent support:
  - if one literal is the negation of the other,
  - if each possible pair of actions that could achieve the two literals is mutually exclusive.

Planning Graph for Heuristic Estimation

The problem is unsolvable if a goal literal is not in the final level of the graph.

The level where a literal goal first arise in the graph give us an estimation of the cost to reach it from the initial state.

If there is a conjunction of goals, we can:

- take the maximum level cost of any of the goals (max-level heuristic);
- take the sum of the level costs of the goals (level sum heuristic);
- take the level where all the literals in the conjunctive goal are in the planning graph without any pair of them being mutually exclusive (the set-level heuristic)

GraphPlan Algorithm

GraphPlan provides a plan directly from the planning graph.

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

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

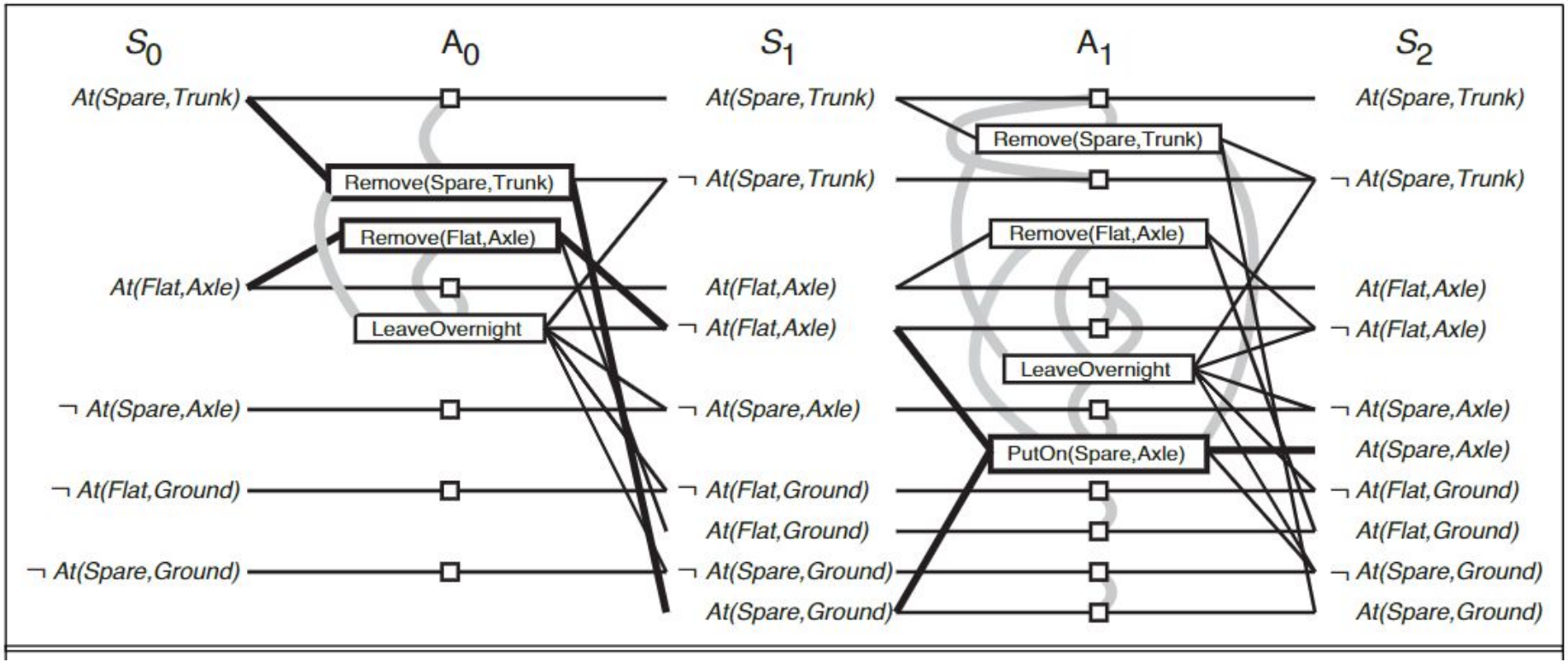
Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

GraphPlan Algorithm

$$
\begin{aligned}
&Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(obj, loc),\\
&\quad \text{PRECOND: } At(obj, loc)\\
&\quad \text{EFFECT: } \neg At(obj, loc) \land At(obj, Ground))\\
&Action(PutOn(t, Axle),\\
&\quad \text{PRECOND: } Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(t, Ground) \land At(t, Axle))\\
&Action(LeaveOvernight,\\
&\quad \text{PRECOND:}\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)\\
&\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))
\end{aligned}
$$

Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

GraphPlan Algorithm



Adaptado de Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).