Planning

Chapter 11

AI – A Modern Approach.

Lecturer:

Srividhya Rajendran

Website for slides download:

http://cseweb.uta.edu/~srajendr

Review

- What is Planning?
- STRIPS Language and Representing planning problems using STRIPS.
- Progression and regression algorithm.
- Heuristics for progression and regression algorithm
- Total ordered planners and partial order planners
- Keywords in partial order planners.

POP Algorithm

- Starts with the definition of initial state, actions, goal test.

- Initial plan contains Start and Finish actions, the ordering constraint $Start \prec Finish$ and no causal links and has all the preconditions in Finish as open preconditions.

POP Algorithm Contd.

- Successor Function: Arbitrarily picks an open precondition p on an action B and generates successor plan.
  - Consistency enforcement:
    - The causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$ are added to the plan.
    - Resolve conflicts between the new causal link and all existing actions. E.g. if action C conflicts with $A \xrightarrow{p} B$ then it is resolved by making C to occur at sometime before action A($C \prec A$) or after action B($B \prec C$)by adding ordering constraints . Add all successor states for either or both if they result in consistent plans.

POP Algorithm Contd.

- Backtrack if an open condition is unachievable or if a conflict is irresolvable.
- Goal test: Checks whether a plan is a solution to the original planning problem by checking if there any open preconditions left. If set of open preconditions set is empty then POP has reached a solution.

POP Example

Agent is at hardware store (HWS). Agent has to buy milk and eggs from supermarket and return home.

Start State:

At(HWS) ∧ ¬ Have (Milk) ∧¬ Have (Eggs) ∧ Sells (SM,Milk) ∧ Sells (SM,Eggs)

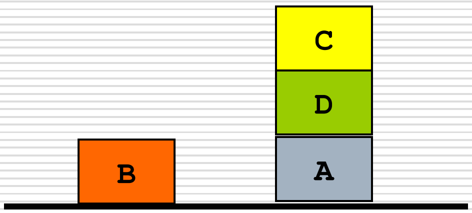
Goal State:

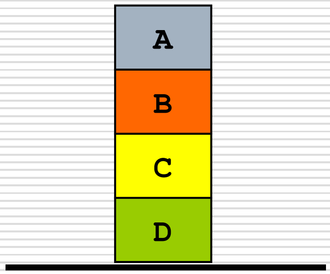
At(Home) ∧ Have (Milk) ∧ Have (Eggs)

(Example using POP)

Blocks World Example using POP

Initial State | **Goal State**





Action :**Move(p,x,y)**

Precond:On(p,x) Λ clear(y) Λ clear(p)

Effect :On(p,y) Λ clear (x) Λ ¬ clear(y) Λ¬ On(p,x))

Op(Action :**Movetotable(p,x)**

Precond:On(p,x) Λ clear(p)

Effect :On(p,Table) Λ clear (x) Λ¬ On(p,x))

(Blocks World Example using POP)

Heuristics for POP

- Count the number of distinct open preconditions.
  - Overestimates: When start state has literal that matches open precondition in finish state.
  - Underestimates: when there is negative interactions between actions.

Heuristics for POP

- Better approach to calculate heuristics:
  - Choose open preconditions that can be satisfied in the fewest number of ways.

Planning Graphs

- Used to achieve better heuristic estimates.
  - A solution can also directly extracted using GRAPHPLAN.
- Consists of a sequence 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.
    - *Literals* = all those that ***could be true*** at that time step, depending upon the actions executed at the preceding time step.
    - *Actions* = all those actions that ***could have their preconditions satisfied*** at that time step, depending on which of the literals actually hold.

Cake Example

Init(Have(Cake))

Goal (Have(Cake) ∧ Eaten(Cake))

Action(**Eat(Cake)**

PRECOND: Have(Cake)

EFFECT:

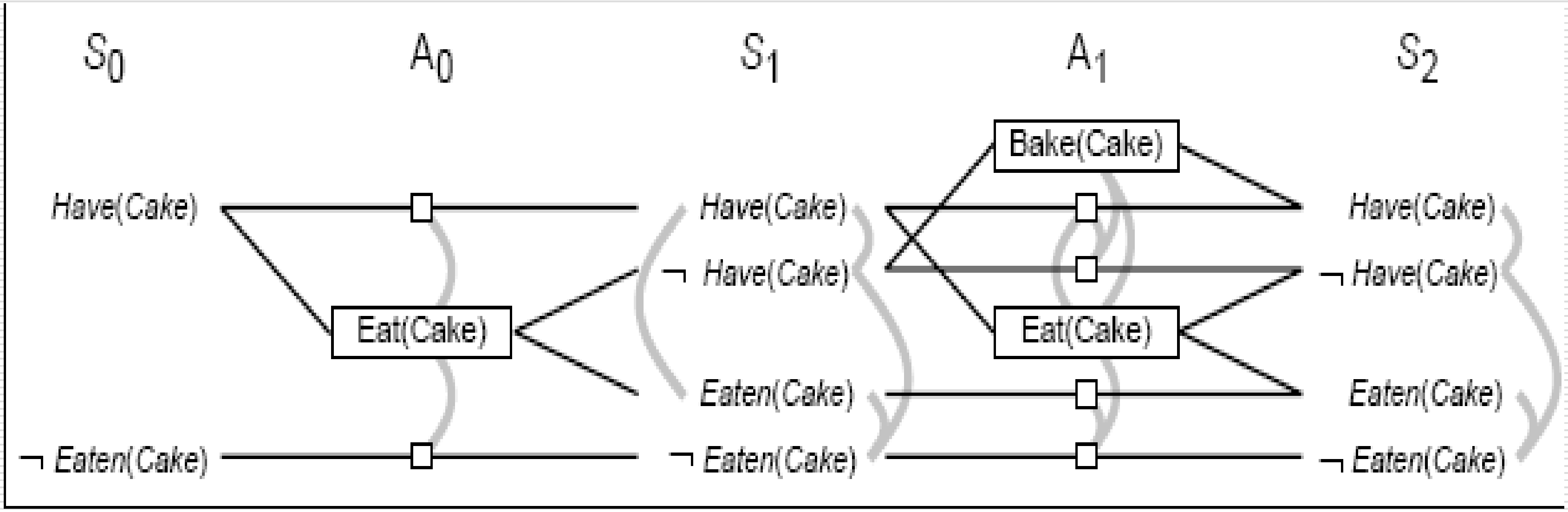
¬Have(Cake)∧Eaten(Cake))

Action(**Bake(Cake)**

PRECOND:Have(Cake)

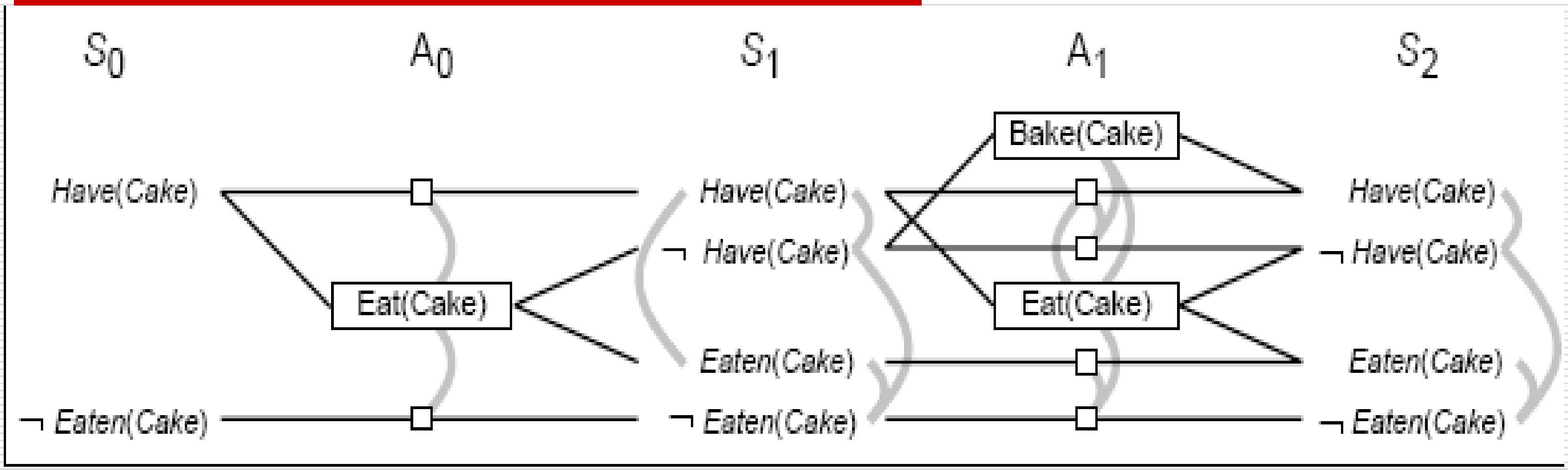
EFFECT: Have(Cake))

Cake Example



- Start at level S0 and determine action level A0 and next level S1.
  - A0 contains all actions whose preconditions are satisfied in the previous level.

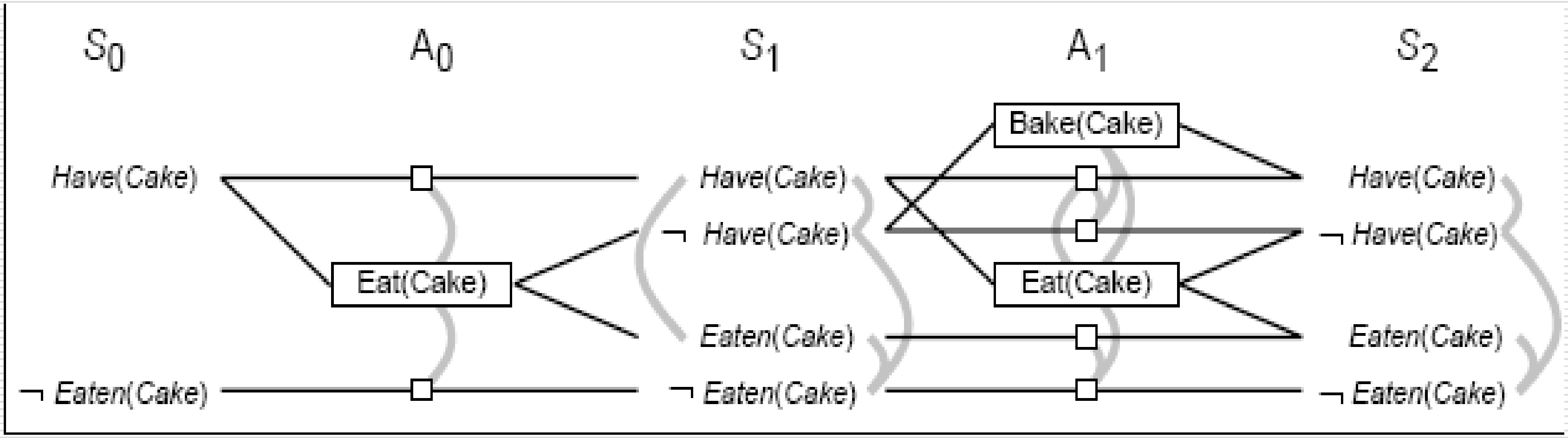
Cake Example



- Connect precondition and effect of actions.
- Inaction is represented by persistence actions.

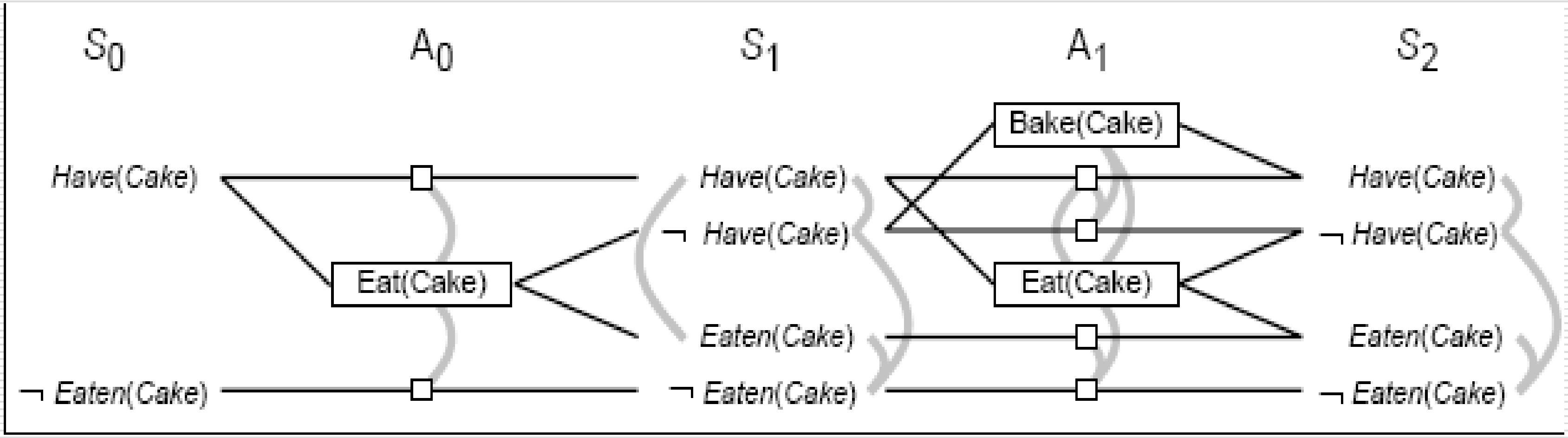
- Level A0 contains the actions that could occur
  - Conflicts between actions are represented by *mutex* links

Cake Example



- 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.

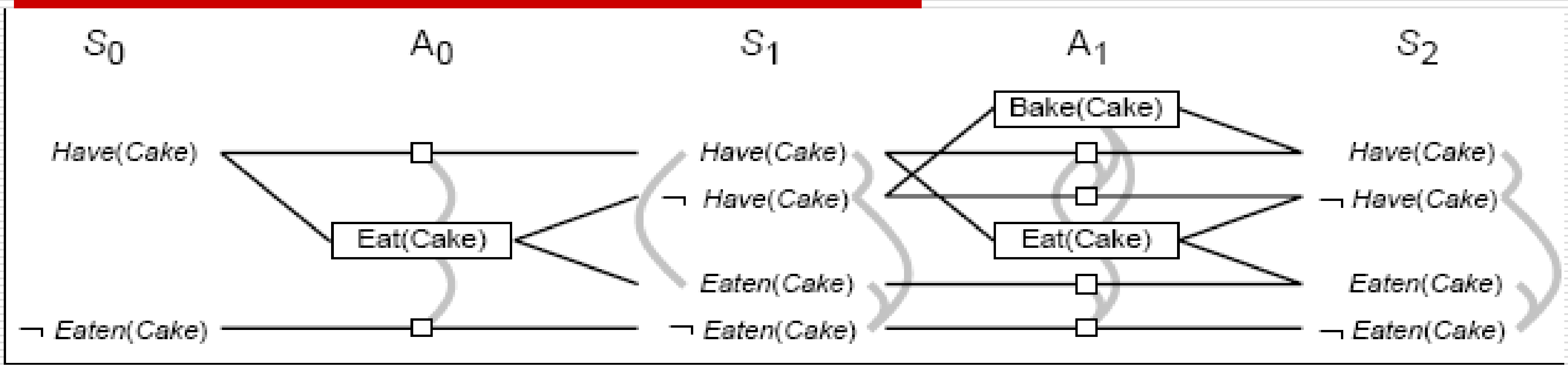
Cake Example



- S1 defines multiple states and the mutex links are the constraints that define this set of states.

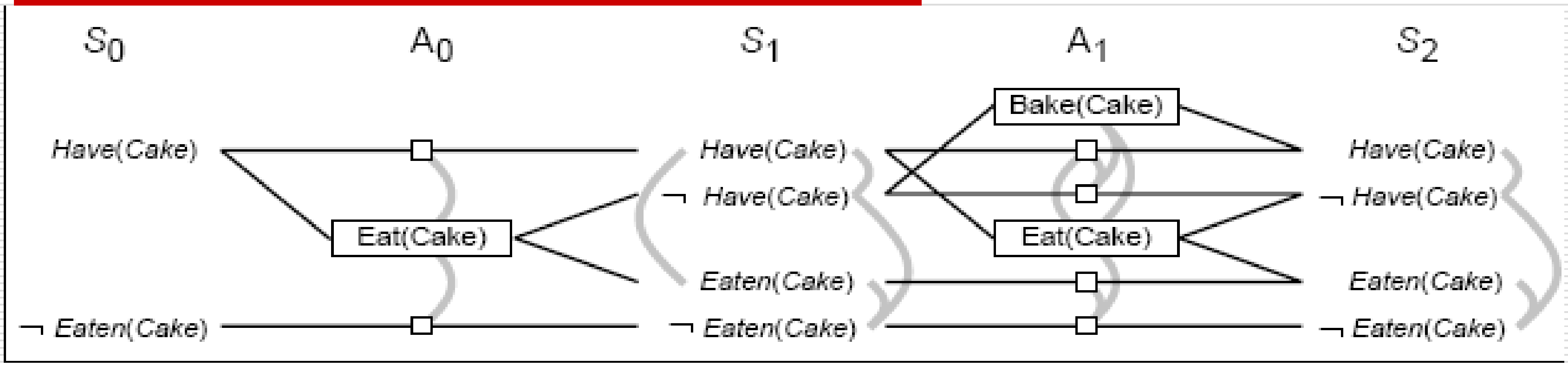
- Continue until two consecutive levels are identical

Cake Example



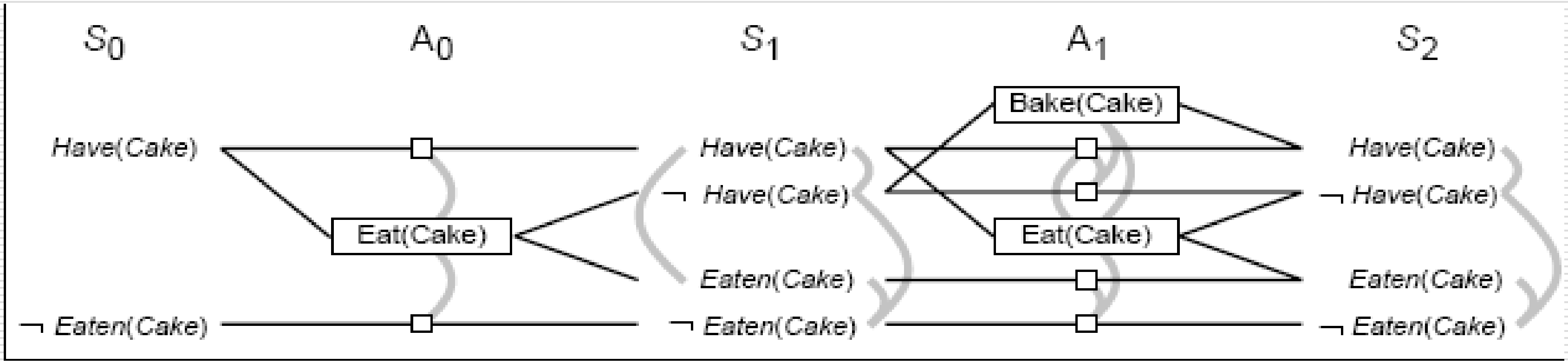
- **A mutex relation holds between two actions when:**
  1. **Inconsistent effects**: one action negates the effect of another.
  2. **Interference:** one of the effects of one action is the negation of a precondition of the other.

Cake Example



- **A mutex relation holds between two actions when:**
  3. **Competing needs:** one of the preconditions of one action is mutually exclusive with the precondition of the other.

Cake Example



- **A mutex relation holds between two literals when (inconsistent support):**
  1. If one is the negation of the other
  2. If each possible action pair that could achieve the literals is mutex.

Planning Graphs for Heuristic Estimation

- A literal that does not appear in the final level of the graph cannot be achieved by any plan.
  - Useful for backward search (cost = inf).
- Level of appearance can be used as cost estimate of achieving any goal literals = *level cost*.
- Small problem: several actions can occur
  - Restrict to one action using serial PG (add mutex links between every pair of actions, except persistence actions).
- Cost of a conjunction of goals? Max-level, sum-level and set-level heuristics.

GraphPlan Algorithm

- extracts a solution directly from the PG

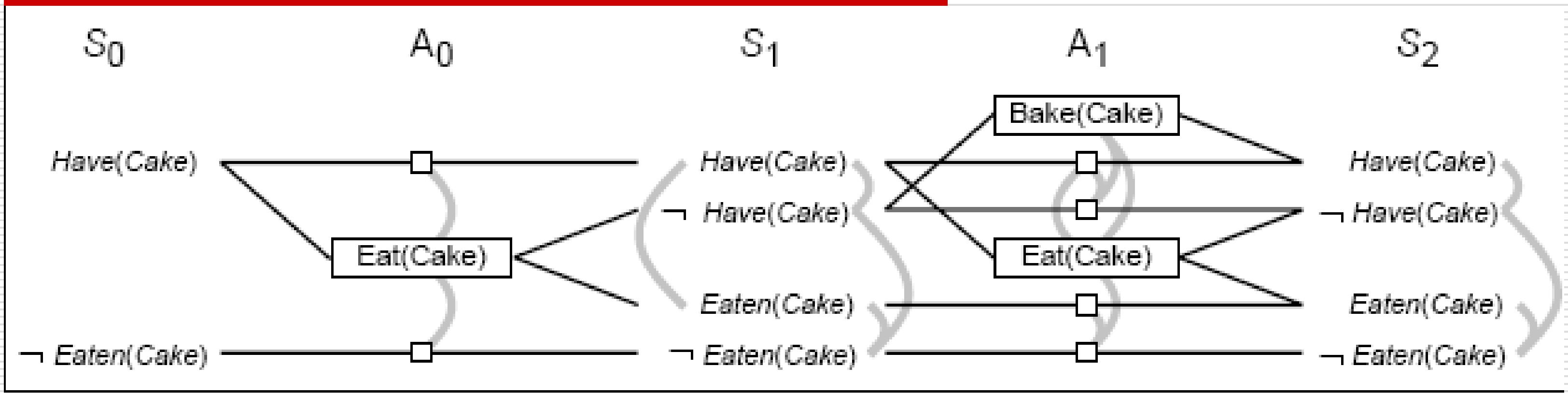
```
function GRAPHPLAN(problem) returns solution or failure
   graph ← INITIAL-PLANNING-GRAPH(problem)
   goals ← GOALS[problem]
   loop do
        if goals all non-mutex in last level of graph then do
           solution ← EXTRACT-SOLUTION(graph, goals,LENGTH(graph))
           if solution ≠ failure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

- **EXTRACT-SOLUTION:**
  - checks whether a plan can be found searching backwards
- **EXPAND-GRAPH:**
  - adds actions for the current and state literals for the next level

Extract Solution

- **A state consists of**
  - a pointer to a level in the planning graph
  - a set of unsatisfied goals
- **Initial state**
  - last level of PG
  - set of goals from the planning problem
- **Actions**
  - select any set of non-conflicting subset of the actions of A_{i-1} that cover the goals in the state
- **Goal**
  - success if level S_0 is reached with such with all goals satisfied

Cake example



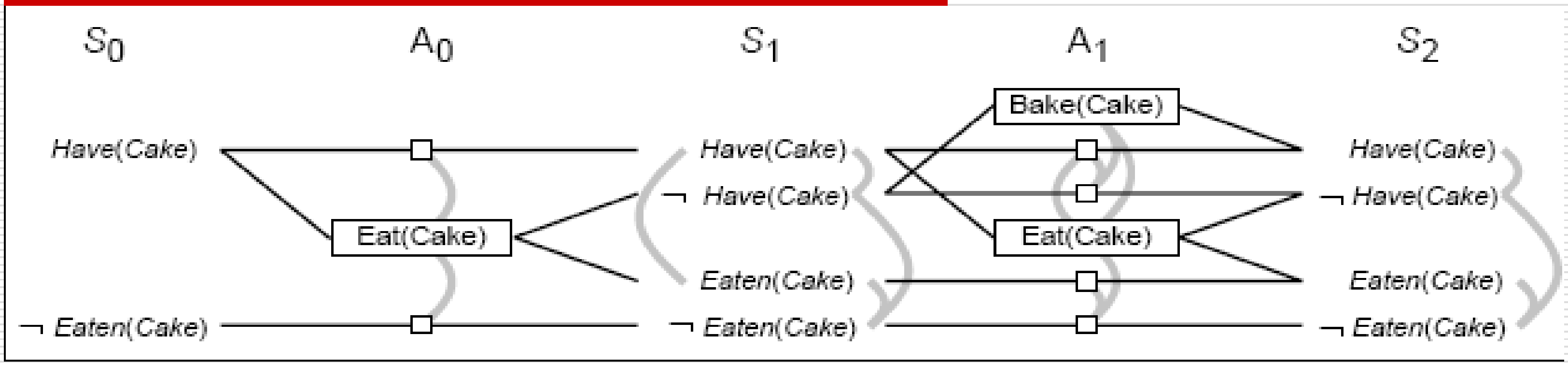
- **Start with Goal state (literals):**

  Have(Cake) ∧ Eaten(Cake) in S_2

- **Only non conflicting Action choices are:** Bake (Cake) ,Persistent action (Eaten (Cake)).

as all the other have mutex relation with respect to either their preconditions or effects.

Cake example



- **Literals at S_1:**

  ¬Have (Cake) ∧ Eaten(Cake).

- **Only action:**

  Eat(Cake)

- **Literals at S0:**

  Have(Cake) ∧ ¬ Eaten(Cake). **(graphplan terminates)**

Planning with Propositional Logic

- Planning can be done by proving theorem in situation calculus.
- Here: test the *satisfiability* of a logical sentence:

$$initial\,state \wedge all\,possible\,action\,descriptions \wedge goal$$

- Sentence contains propositions for every action occurrence.
  - A model will assign true to the actions that are part of the correct plan and false to the others
  - An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that the goal is true.
  - If the planning is unsolvable the sentence will be unsatisfiable.