Planning in Artificial Intelligence

The intelligent way to do things

COURSE: CS60045

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

USING PLANNING GRAPHS

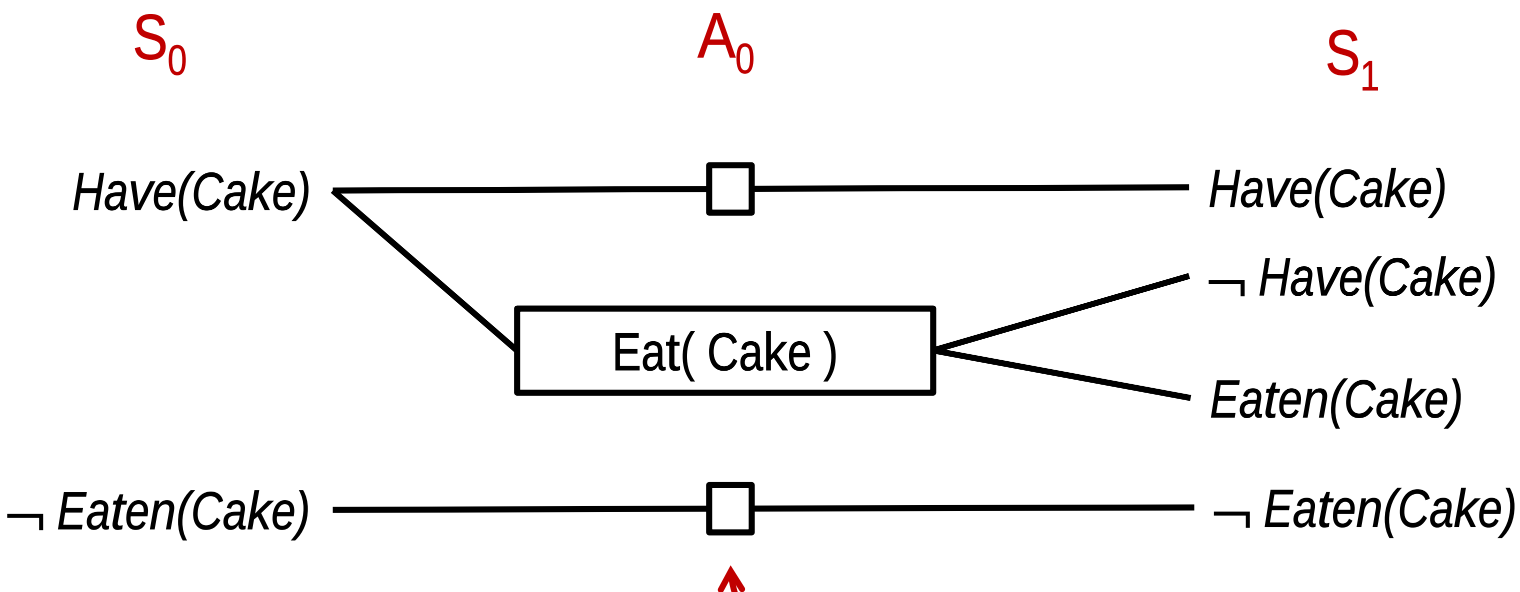
GraphPlan and SATPlan

Planning Graph

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

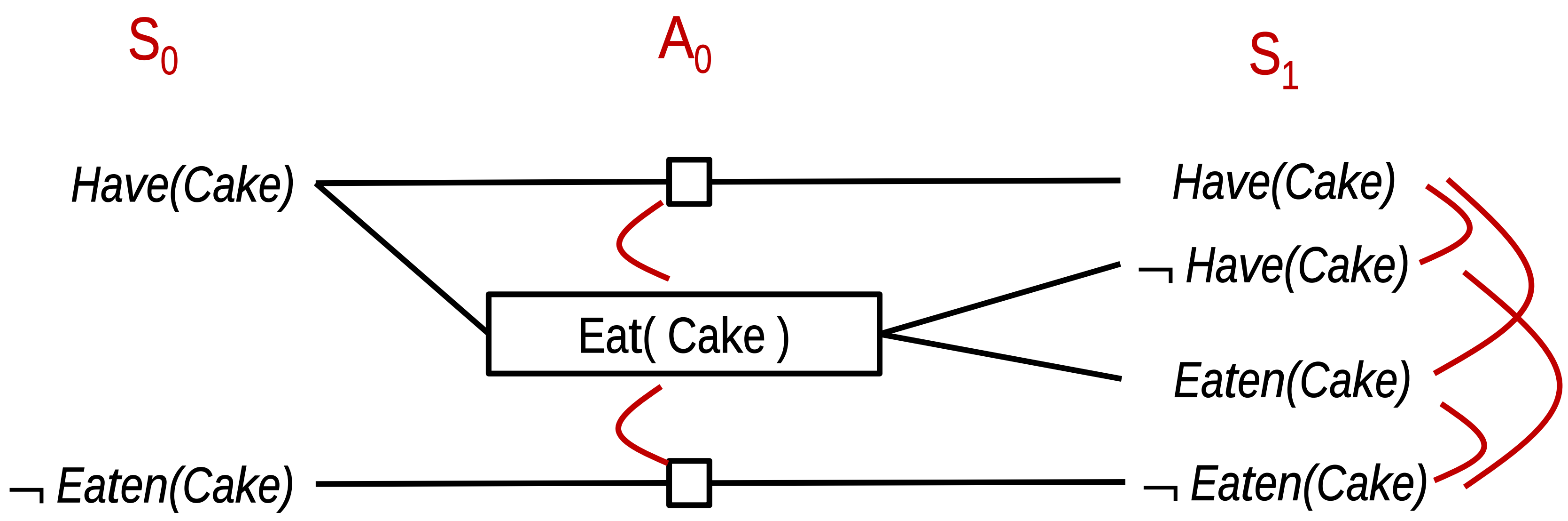
Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Op(ACTION: Bake(Cake),
PRECOND: ¬Have(Cake),
EFFECT: Have(Cake))



Persistence action
(carries over a predicate to the next world)

Mutex Links in a Planning Graph



Mutual exclusion among actions

Mutual exclusion among derived predicates

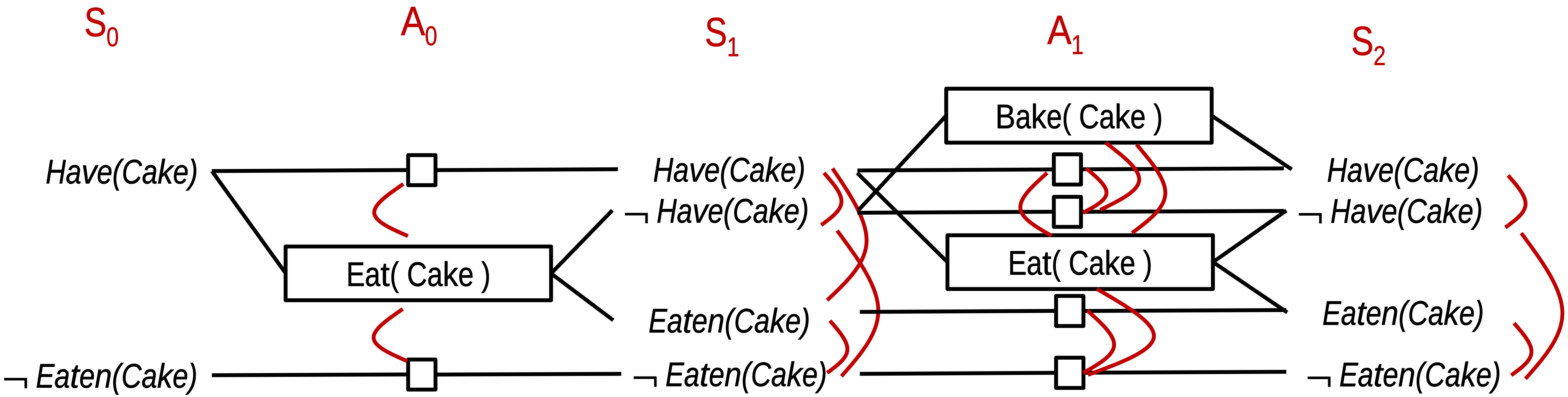
Planning Graphs

- Consists of a sequence of levels that correspond to time steps in the plan
- Each level contains a set of actions and a set of literals that *could* be true at that time step depending on the actions taken in previous time steps
- For every +ve and –ve literal C, we add a *persistence action* with precondition C and effect C

Planning Graph

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

Op(ACTION: Bake(Cake),
PRECOND: ¬Have(Cake),
EFFECT: Have(Cake))



Start: Have(Cake)
Finish: Have(Cake) $\wedge$ Eaten(Cake)

In the world S_2 the goal predicates exist without mutexes, hence we need not expand the graph any further

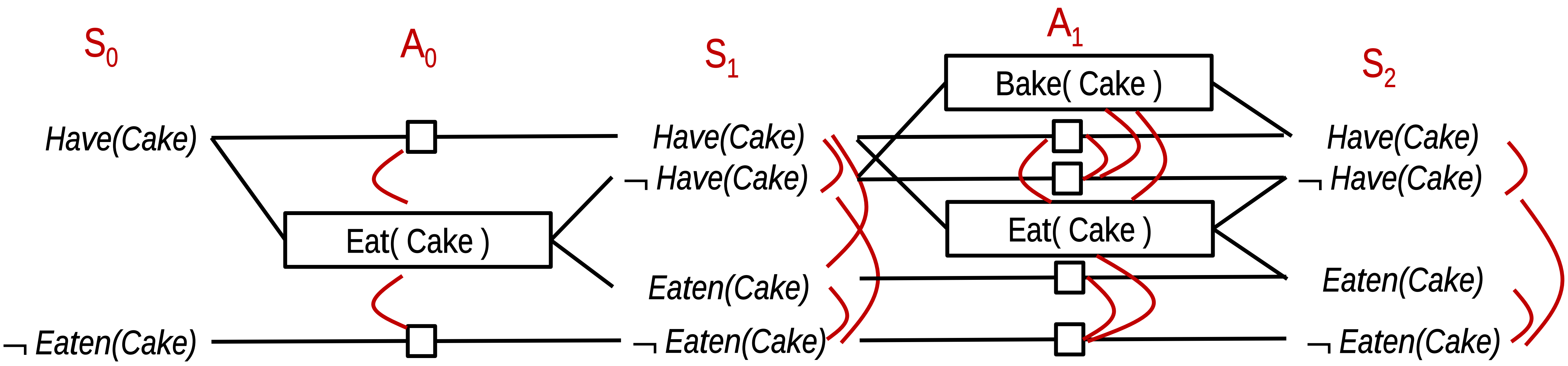
Mutex Actions

- Mutex relation exists between two actions if:
  - Inconsistent effects – one action negates an effect of the other

    Eat(Cake) causes ¬ *Have(Cake)* and Bake(Cake) causes *Have(Cake)*
  - Interference – one of the effects of one action is the negation of a precondition of the other

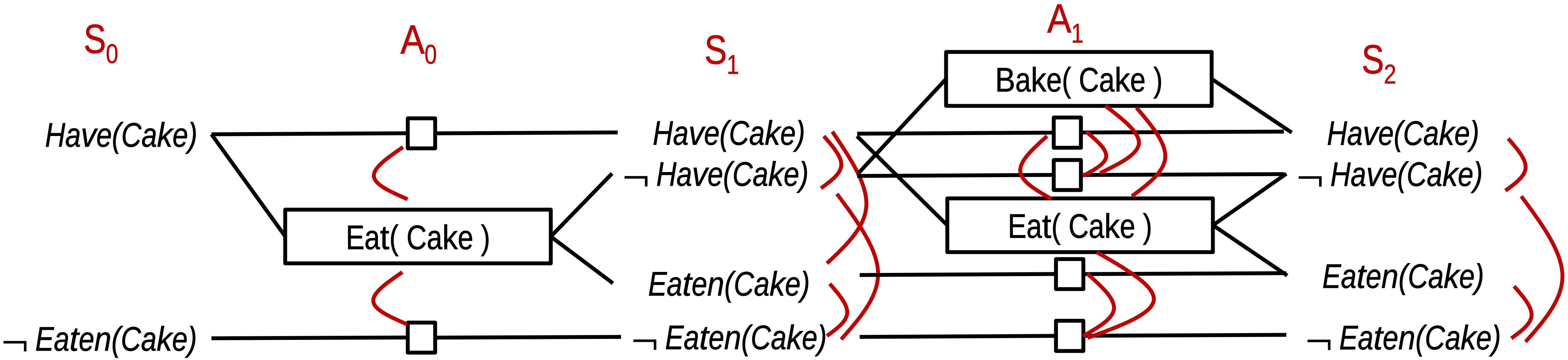
    Eat(Cake) causes ¬ *Have(Cake)* and the persistence of *Have(Cake)* needs *Have(Cake)*
  - Competing needs – one of the preconditions of one action is mutually exclusive with a precondition of the other

    Bake(Cake) needs ¬ *Have(Cake)* and Eat(Cake) needs *Have(Cake)*



Mutex Literals

- Mutex relation exists between two literals if:
  - One is the negation of the other, or
  - Each possible pair of actions that could achieve the two literals is mutually exclusive (inconsistent support)

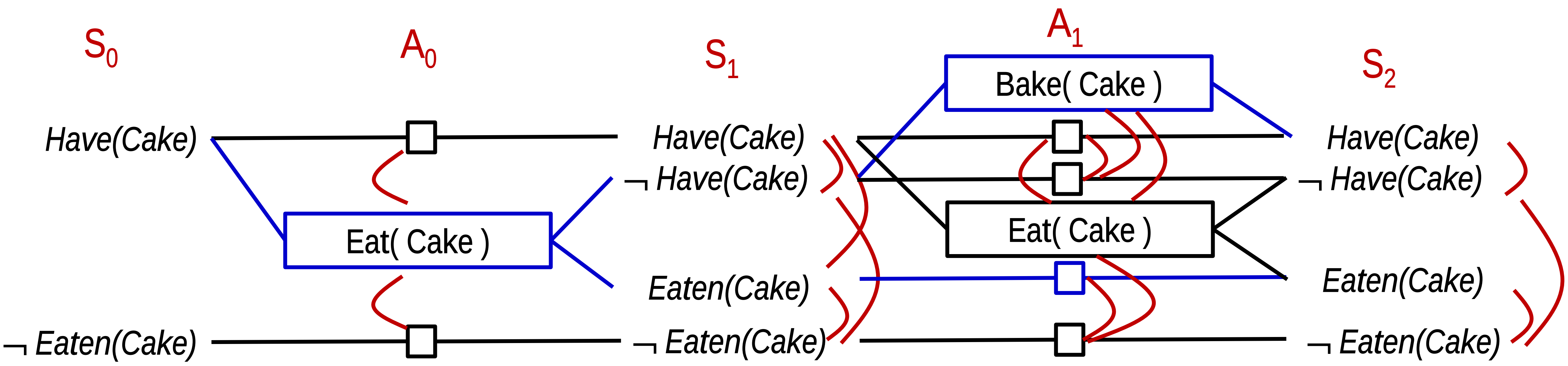


Function GraphPLAN(problem)

```
    // returns solution or failure
    graph ← Initial-Planning-Graph( problem )
    goals ← Goals[ problem ]
    do
        if goals are all non-mutex in last level of graph then do
            solution ← Extract-Solution( graph )
            if solution ≠ failure then return solution
            else if No-Solution-Possible (graph )
                then return failure
        graph ← Expand-Graph( graph, problem )
```

Finding the plan

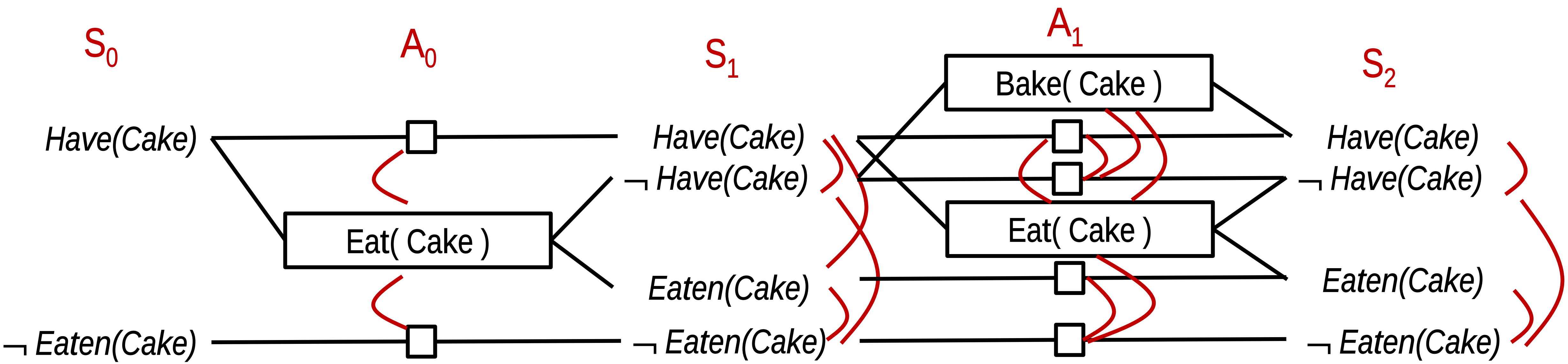
- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.
- The plan is shown in blue below



Termination of GraphPLAN when no plan exists

- Literals increase monotonically
- Actions increase monotonically
- Mutexes decrease monotonically

This guarantees the existence of a fixpoint



Planning with Propositional Logic

- The planning problem is translated into a CNF satisfiability problem
- The goal is asserted to hold at a time step T, and clauses are included for each time step up to T.
- If the clauses are satisfiable, then a plan is extracted by examining the actions that are true.
- Otherwise, we increment T and repeat

Example

Aeroplanes P_1 and P_2 are at SFO and JFK respectively. We want P_1 at JFK and P_2 at SFO

Initial: $At(P_1, SFO)^0 \wedge At(P_2, JFK)^0$

Goal: $At(P_1, JFK) \wedge At(P_2, SFO)$

Action: $At(P_1, JFK)^1 \Leftrightarrow [\, At(P_1, JFK)^0 \wedge \neg\, (Fly(P_1, JFK, SFO)^0 \wedge At(P_1, JFK)^0)\,]$

$$\vee\, [\, At(P_1, SFO)^0 \wedge Fly(P_1, SFO, JFK)^0\,]$$

Check the satisfiability of:

initial state $\wedge$ *successor state axioms* $\wedge$ *goal*

Additional Axioms

Precondition Axioms:

$$Fly(P_1, JFK, SFO)^0 \Rightarrow At(P_1, JFK)^0$$

Action Exclusion Axioms:

$$\neg\,(Fly(P_2, JFK, SFO)^0 \wedge Fly(P_2, JFK, LAX)^0)$$

State Constraints:

$$\forall\, p, x, y, t \;\; (x \neq y) \Rightarrow \neg\,(At(p, x)^t \wedge At(p, y)^t)$$

SATPlan

Function SATPlan(problem, T_{max})

// *returns solution or failure*

for T = 0 to T_{max} do

cnf, mapping ← Trans-to-SAT(*problem*, T)

assignment ← SAT-Solver(*cnf*)

if *assignment* is not NULL then

return Extract-Solution(*assignment, mapping*)

return *failure*

Further Readings

- Heuristic Search Planning

- Planning with Temporal Goals
- Planning under Adversaries
- Multi-agent Planning
- Planning in Continuous State Spaces

- Planning with Reinforcement Learning

Explainable AI Planning (XAIP)

Enables you to seek explanations from the planner.

- Why did you do that?
- And why didn't you do something else (which I would have chosen)?
- Why is what you propose better / cheaper / safer than what I would have done?
- Why can't you do that?
- Why do I need to backtrack (and replan) at this point?
- Why do I not need to replan at this point?

Exercise-1

Start: At(Flat, Axle) ∧ At(Spare, Trunk)

Goal: At(Spare, Axle)

Op(ACTION: Remove(Spare, Trunk),
PRECOND: At(Spare, Trunk),
EFFECT: At(Spare, Ground)
∧ ¬ At(Spare, Trunk))

Op(ACTION: Remove(Flat, Axle),
PRECOND: At(Flat, Axle),
EFFECT: At(Flat, Ground)
∧ ¬ At(Flat, Axle))

Op(ACTION: PutOn(Spare, Axle),
PRECOND: At(Spare, Ground)
∧ ¬ At(Flat, Axle),
EFFECT: At(Spare, Axle)
∧ ¬ At(Spare, Ground))

Op(ACTION: LeaveOvernight,
PRECOND:
EFFECT: ¬ At(Spare, Ground)
∧ ¬ At(Spare, Axle)
∧ ¬ At(Spare, Trunk)
∧ ¬ At(Flat, Ground)
∧ ¬ At(Flat, Axle))

Use the partial order planning algorithm to develop a plan for this domain.

Exercise-2

Consider the following list of actions.

- The initial world is defined by $\neg$ Have(Pizza) $\wedge \neg$ Have(Cake).
- The planning goal is: Gastric $\wedge$ Toothache $\wedge \neg$ Hungry.

Draw the planning graph after two levels of actions and indicate (with justification) whether we already have a plan. Your planning graph should clearly specify the mutex relations between the actions and the facts.

ACTION	PRECOND	EFFECT
Bake(x)	$\neg$ Have(x)	Have(x)
Eat-Pizza	Have(Pizza) $\wedge \neg$ Have(Cake)	Gastric $\wedge \neg$ Hungry
Eat-Cake	Have(Cake) $\wedge \neg$ Have(Pizza)	Toothache $\wedge \neg$ Hungry
Eat-Both	Have(Cake) $\wedge$ Have(Pizza)	Gastric $\wedge$ Toothache

Exercise-3 (No, you don't need to read the book, nor watch the movies to solve this one)

Lord Voldemort wishes to acquire the *elder wand*, the *resurrection stone*, and the *invisibility cloak*. There are actions by which he wishes to get these, but the actions also have other side effects. He has written down the actions as follows:

Op(ACTION: GetWand,	PRECOND: At(x),	EFFECT: Have(wand) $\wedge$ $\neg$Happy)
Op(ACTION: GetStone,	PRECOND: At(x),	EFFECT: Have(stone) $\wedge$ Safe)
Op(ACTION: StealCloak,	PRECOND: At(x),	EFFECT: Have(cloak) $\wedge$ Invisible $\wedge$ Happy)
Op(ACTION: BuyCloak,	PRECOND: At(x),	EFFECT: Have(cloak) $\wedge$ $\neg$Invisible $\wedge$ $\neg$Safe)
Op(ACTION: Start,		EFFECT: At(Hogwarts))
Op(ACTION: Finish,	PRECOND: Have(wand) $\wedge$ Have(stone) $\wedge$ Have(cloak))	

1. Voldemort has decided to use the GraphPlan algorithm to choose his plan. Draw the planning graph after one iteration, clearly indicating all the mutex links.
2. Is any further iteration necessary? Explain.
3. Will GraphPlan terminate with a plan in this case? If so, draw the plan. If not, explain why.