GraphPLAN and SATPlan

COURSE: CS40002

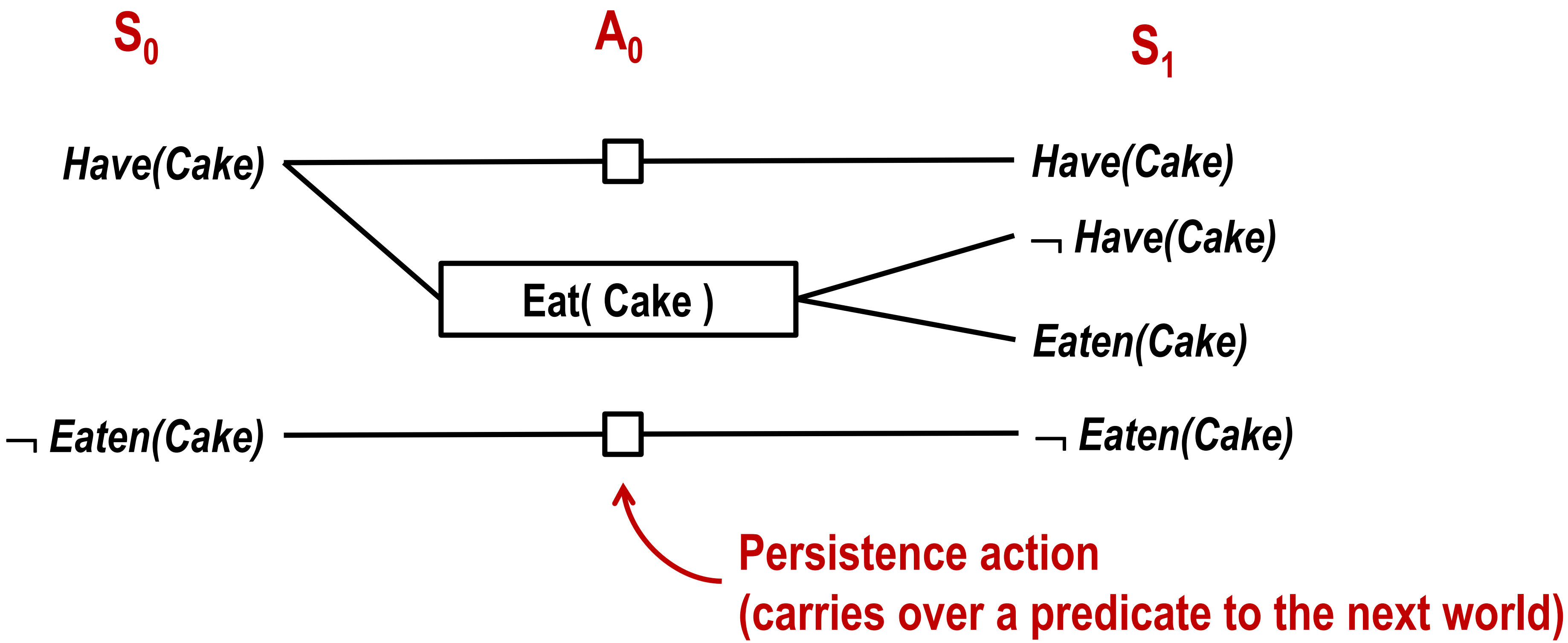
Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Planning Graph

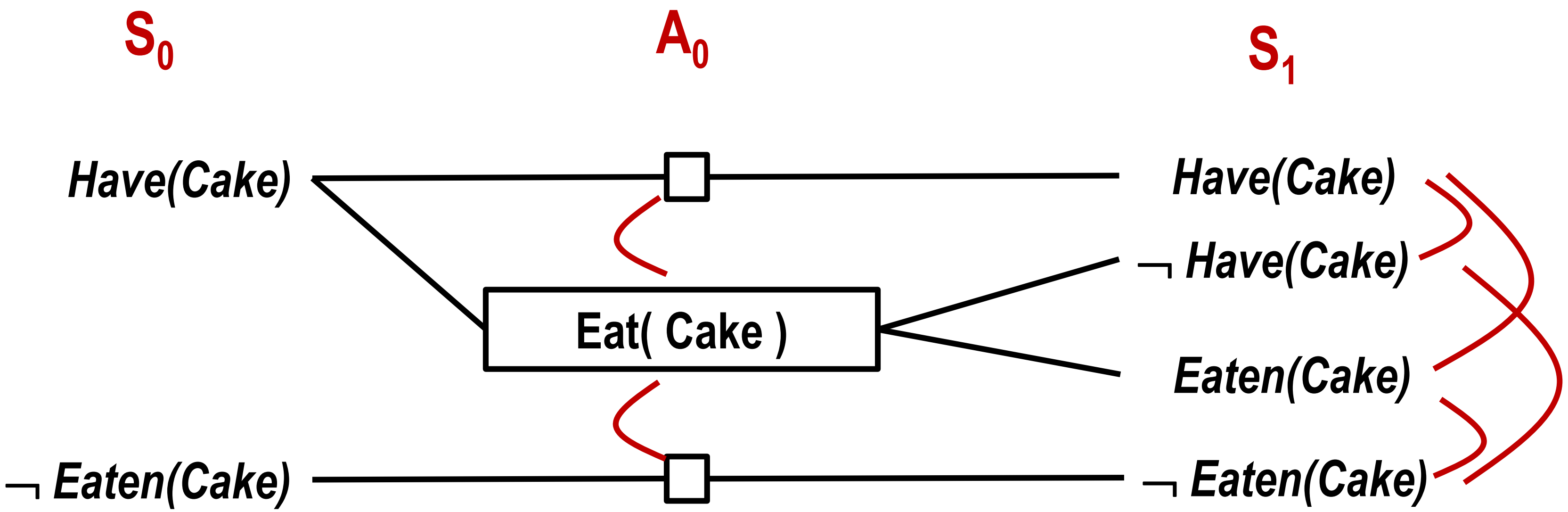
Op(**ACTION**: Eat(Cake),
PRECOND: Have(Cake),
EFFECT: Eaten(Cake) ∧ ¬Have(Cake))

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

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



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

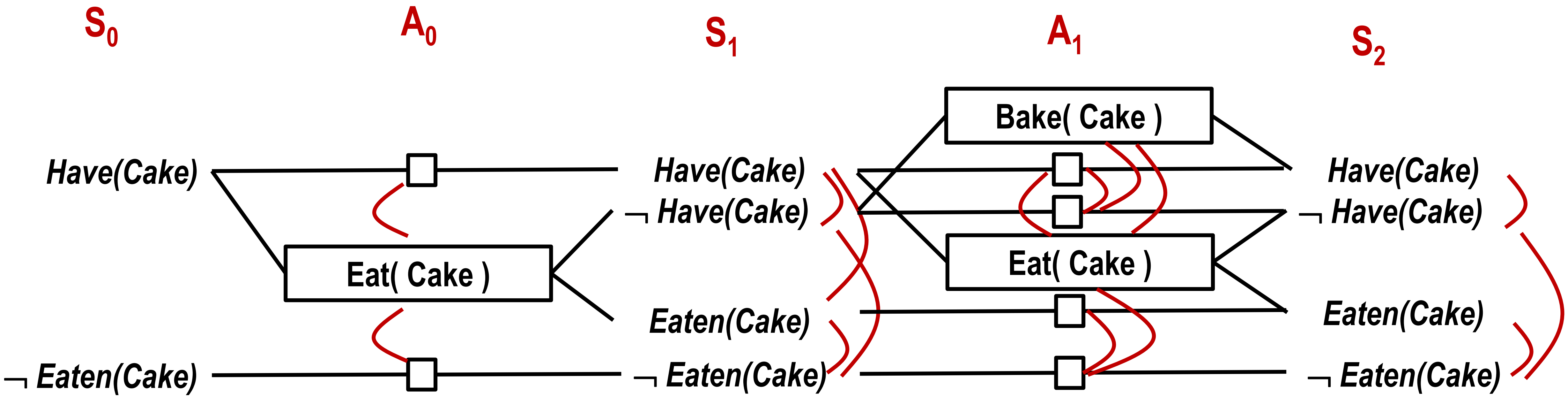
- Each level contains a set of actions and a set of literals that *could* be true at that time step depending on the actions taken in previous time steps

- 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) ∧ ¬Have(Cake))

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



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

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

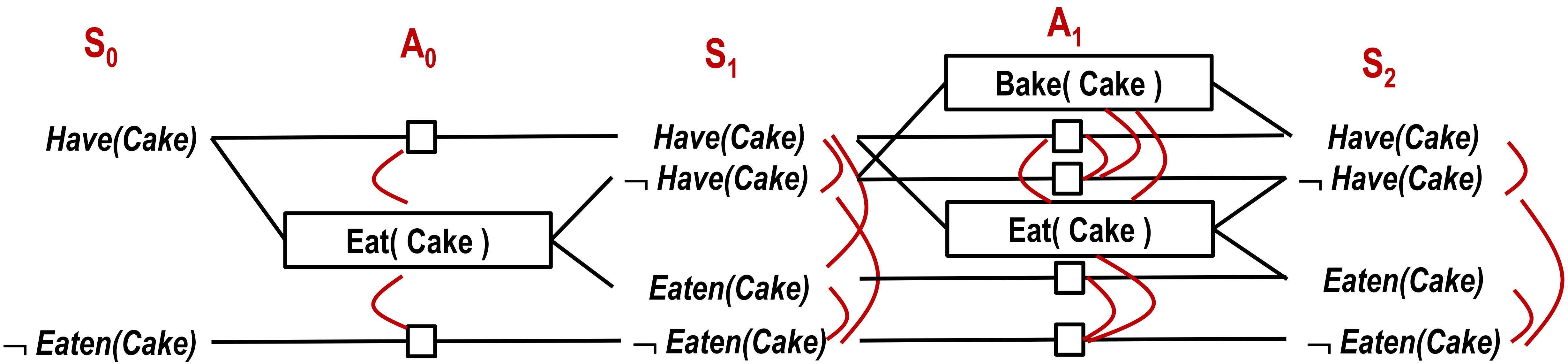
Mutex Actions

- **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

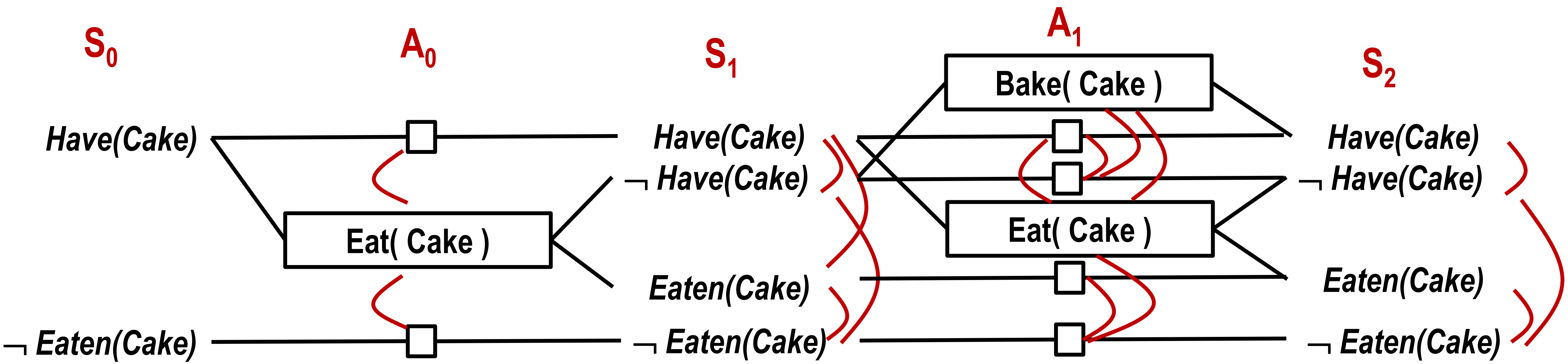
    Eat(Cake) causes ¬ *Have(Cake)* and the persistence of *Have(Cake)* needs *Have(Cake)*
  - **Competing needs** – one of the preconditions of one action is mutually exclusive with a precondition of the other

    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)

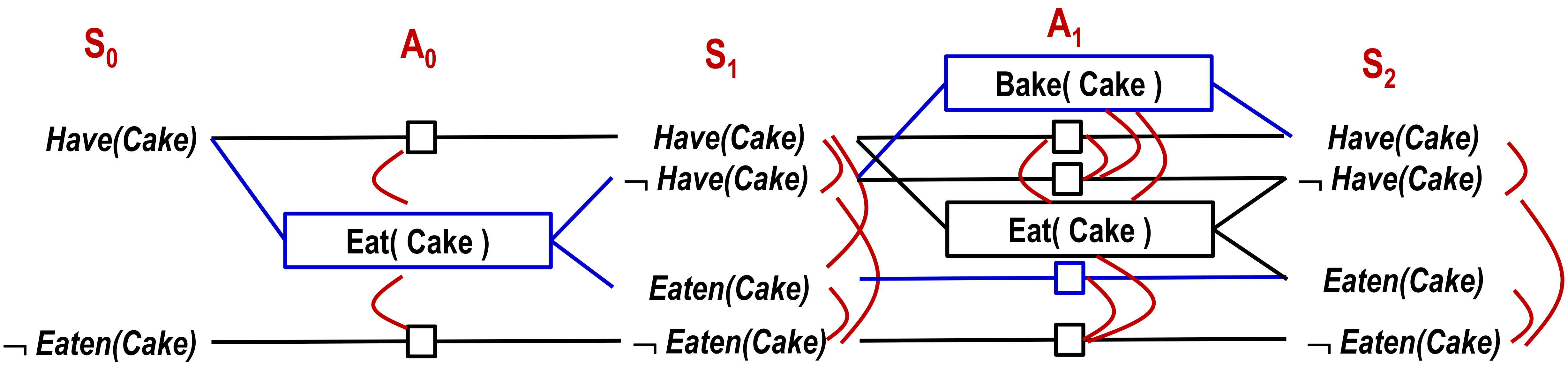


Function GraphPLAN(problem)

```
// 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

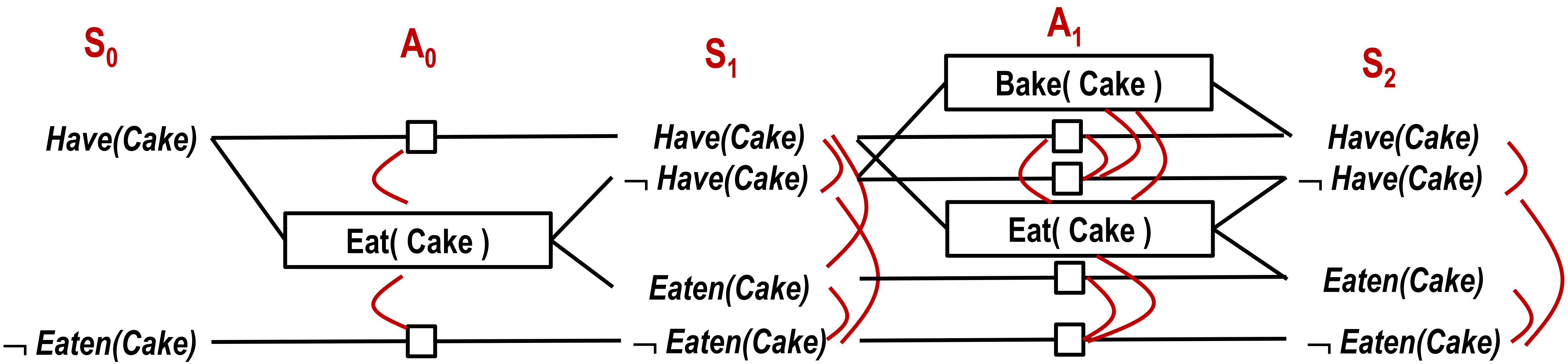
- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness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



Exercise

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))

Symbolic Representation of State Spaces

- States are represented by state vectors: $\langle x_1, x_2, \ldots, x_k \rangle$
- Sets of states can be represented by formulae over the state variables
  - Consider the following set of states:

    $\langle 0\ 1\ 1 \rangle$

    $\langle 0\ 0\ 1 \rangle$

    $\langle 0\ 1\ 0 \rangle$

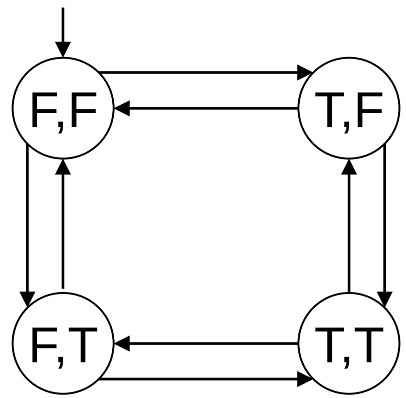
    $\langle 0\ 0\ 0 \rangle$

    $\langle 1\ 1\ 1 \rangle$

    $\langle 1\ 0\ 0 \rangle$
  - The set of states can be represented as a formula: $\neg x_1 \vee (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3)$

Symbolic Search

Variables: x, y: boolean



Set of states:

Q = {(F,F), (F,T), (T,F), (T,T)}

Initial condition:

$Q_0 \equiv \neg x \land \neg y$

Transition relation (negates one variable at a time):

$R \equiv [\ (x' = \neg x) \land (y' = y)\ ] \lor [\ (x' = x) \land (y' = \neg y)\ ]$ (= means $\leftrightarrow$)

x' is the next value of x, and y' is the next value of y

The Simple Example Contd.

Suppose $p \equiv x \wedge y$ defines the goal states.

Our options:

FORWARD SEARCH: Start from the initial state and search for paths to the bad states.

BACKWARD SEARCH: Start from the bad states and work backwards to see whether we reach an initial state.

CORE STEP IN BACKWARD SEARCH: *Find the states that have a successor satisfying p*

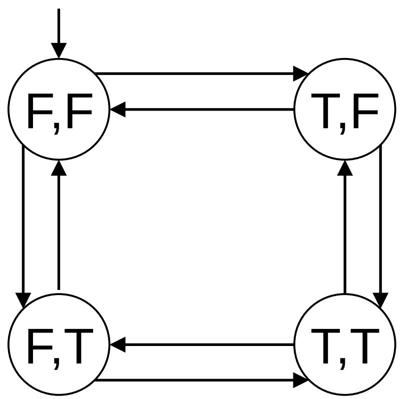
$\text{Pre-Image}(p) \equiv \exists V' \; R \wedge (x' \wedge y')$

$\equiv \exists V' \; [(x' = \neg x \wedge y' = y) \vee (x' = x \wedge y' = \neg y)] \wedge (x' \wedge y')$

$\equiv \exists V' \; [(x' = \neg x \wedge y' = y) \wedge (x' \wedge y')] \vee [(x' = x \wedge y' = \neg y) \wedge (x' \wedge y')]$

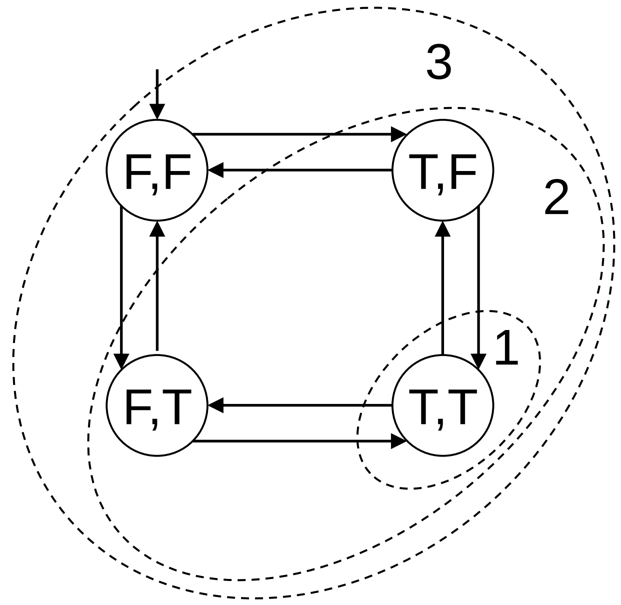
$\equiv \exists V' \; [\neg x \wedge y \wedge x' \wedge y'] \vee [x \wedge \neg y \wedge x' \wedge y']$

$\equiv [\neg x \wedge y] \vee [x \wedge \neg y]$



This formula represents the set of states {(F,T), (T,F)}, which is the set of states having a successor satisfying p

The Simple Example Contd.



Suppose $p \equiv x \wedge y$ defines the set of bad states.

Pre-Image(p) $\equiv [\neg x \wedge y] \vee [x \wedge \neg y]$

FIXPOINT COMPUTATION for BACWARD REACHABILITY

$Z_0 = p$

$Z_1 = Z_0 \vee$ Pre-Image(Z_0)

$Z_2 = Z_1 \vee$ Pre-Image(Z_1)

… and so on, until we have $Z_k = Z_{k-1}$ for some *k*. We call it Z*

Then Z_k is a Boolean formula that represents the set of states that can reach the bad states.

The goal state is reachable if $Q_0 \wedge Z_k$ is satisfiable.

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

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
```