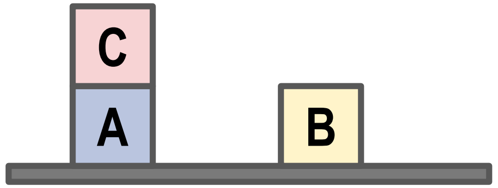
# Planning in Artificial Intelligence

*The intelligent way to do things*

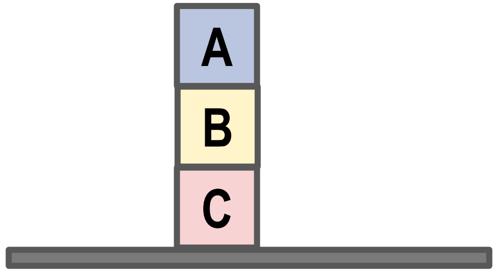
COURSE: CS60045

**Pallab Dasgupta**
**Professor,**
**Dept. of Computer Sc & Engg**

# Blocks World




Initial State




Target State

**Predicates describing the initial state:**
On(C, A), On(A, Table),
On(B, Table),
Clear(C), Clear(B)

**Predicates describing the target state:**
On(A, B), On(B, C)

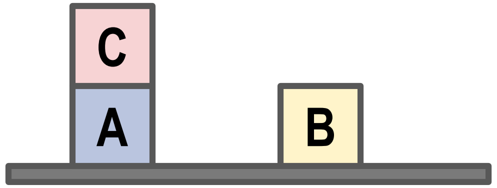
**ACTIONS:**

**Move(X, Y)**
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

**Move(X, Table)**
Precond: Clear(X)
Effect: On(X, Table)

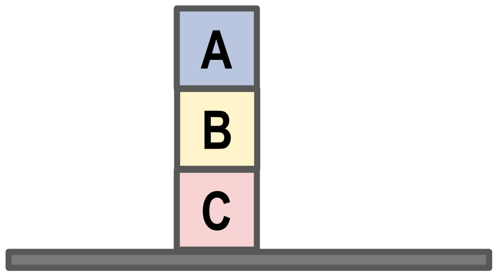
**The planning task is to determine the actions for reaching the target state from the initial state.**

# Choosing Actions




On(C, A), On(A, Table), On(B, Table), Clear(C), Clear(B)

On(A, B), On(B, C)




**ACTIONS:**

**Move(X, Y)**
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

**Move(X, Table)**
Precond: Clear(X)
Effect: On(X, Table)

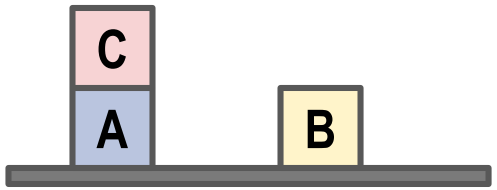
- We can move C to the table
  - This achieves none of the goal predicates
- We can move C to top of B
  - This achieves none of the goal predicates
- We can move B to top of C
  - This achieves On(B, C)

# Partial Solutions

**ACTIONS:**

**Move(X, Y)**
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

**Move(X, Table)**
Precond: Clear(X)
Effect: On(X, Table)

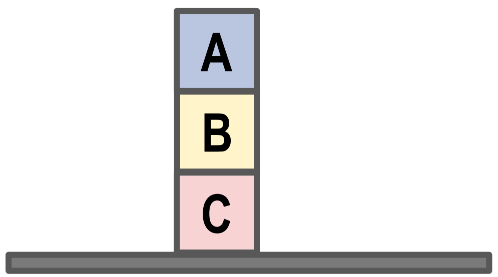



On(C, A), On(A, Table), On(B, Table), Clear(C), Clear(B)

Clear(C), Clear(B)

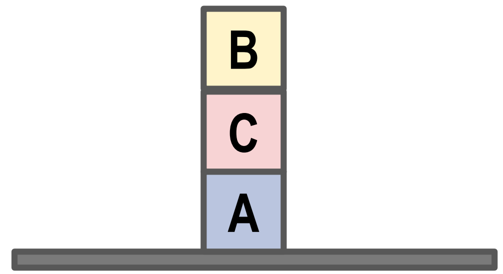
Move(B, C)

On(A, B), On(B, C)



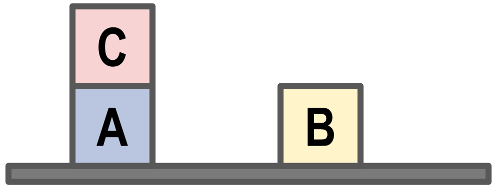

We use Move(B, C) to achieve the sub-goal, On(B, C).

But if we apply this move at the beginning, we get:




Which is not what we want !!

# Partial Solutions




On(C, A), On(A, Table), On(B, Table), Clear(C), Clear(B)

Clear(C)

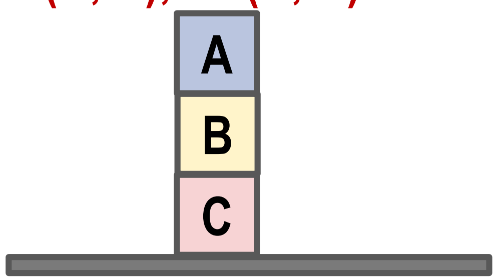
Move(C, Table)

Clear(A), On(C, Table)

Clear(A), Clear(B)

Move(A, B)

On(A, B), On(B, C)

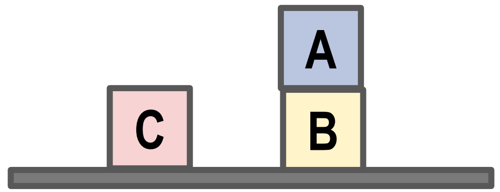



**ACTIONS:**

Move(X, Y)
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

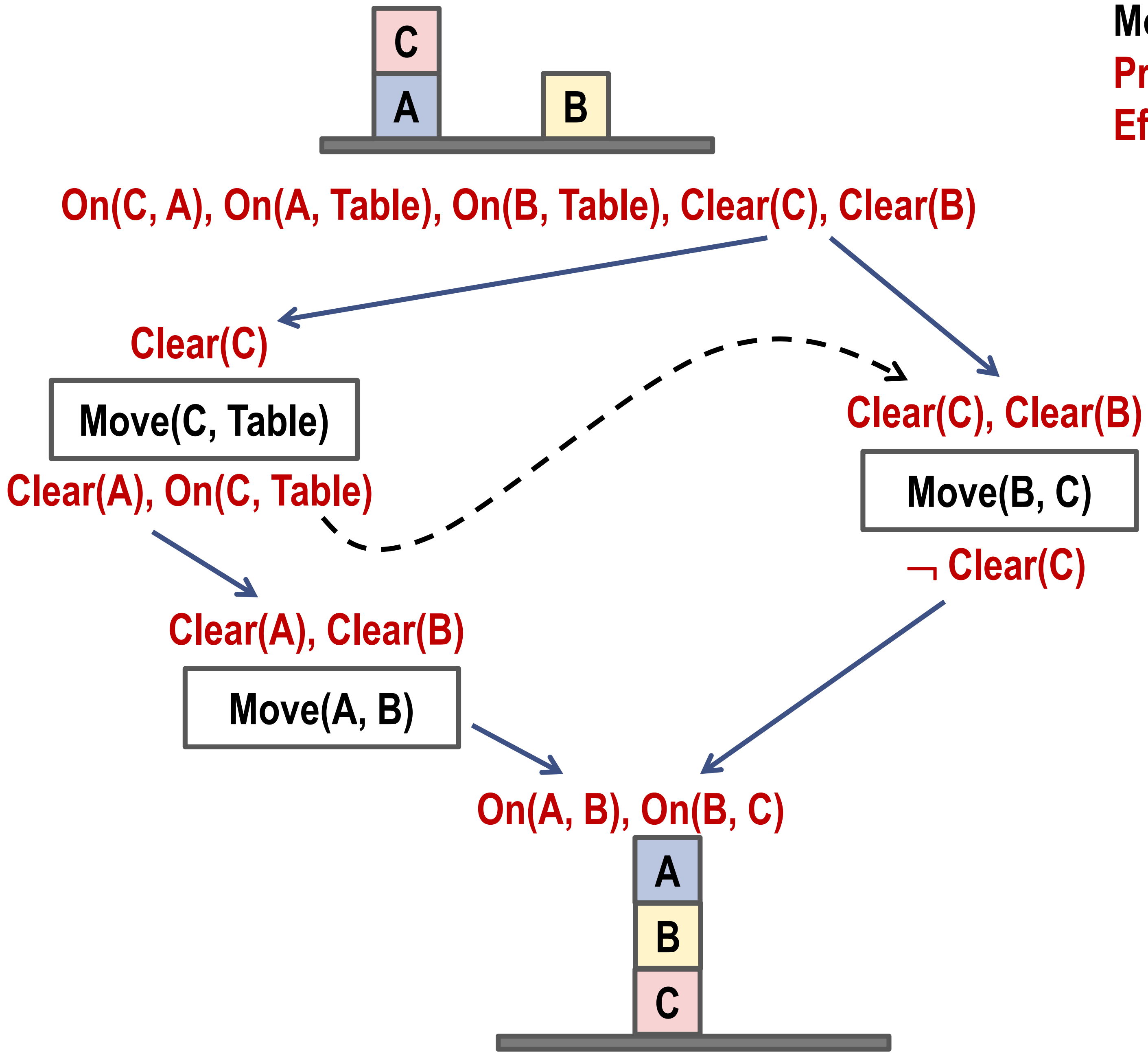
Move(X, Table)
Precond: Clear(X)
Effect: On(X, Table)

The sub-goal On(A, B) is achieved by moving C to the table and then moving A to top to B. But this gives us:




But this too is not what we want !!

# Ordering Partial Solutions




**ACTIONS:**

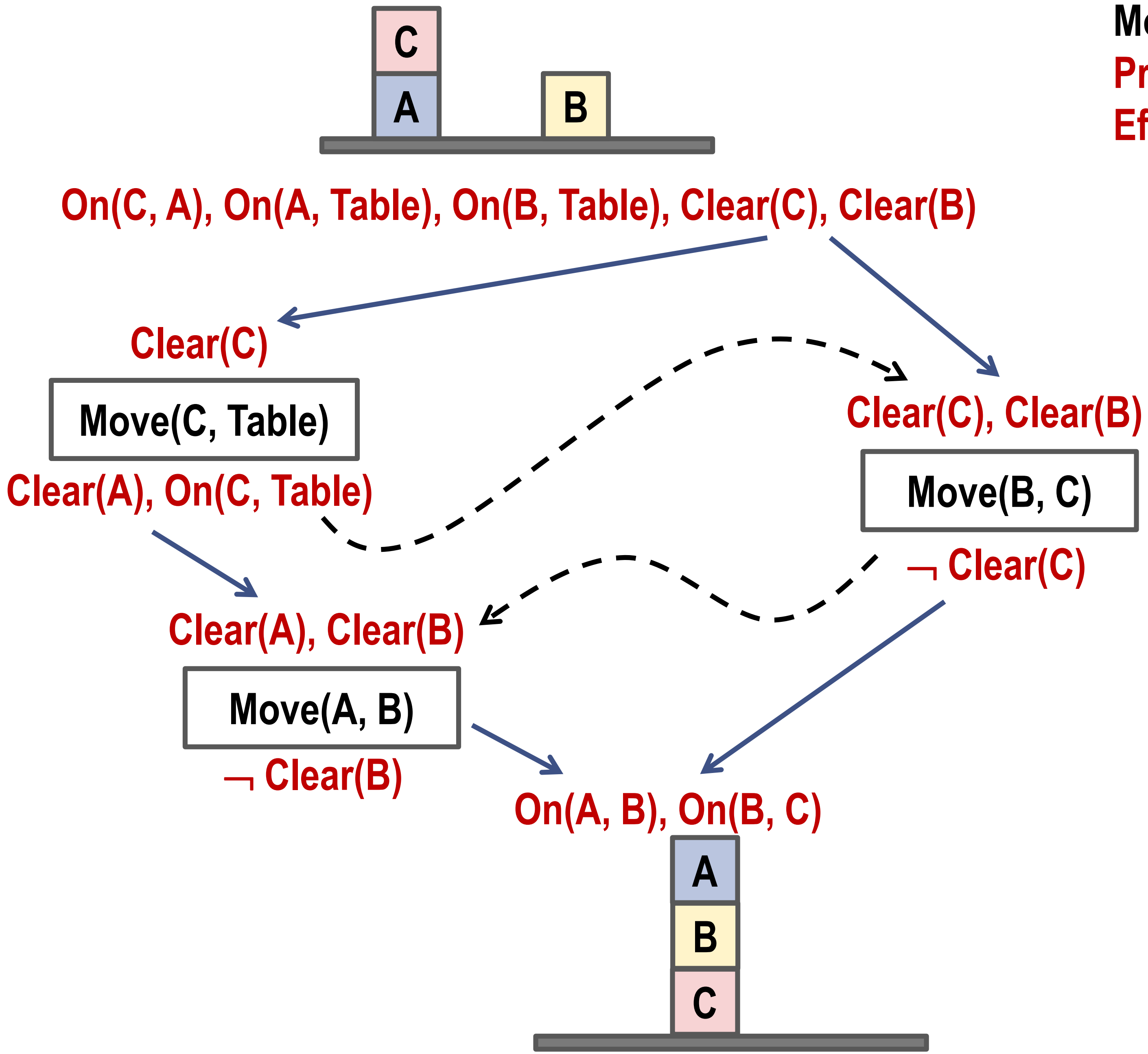
**Move(X, Y)**
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

**Move(X, Table)**
Precond: Clear(X)
Effect: On(X, Table)

Move(B, C) removes the Clear(C) predicate which is essential for Move(C, Table). Hence Move(C, Table) must precede Move(B, C).

Can Move(B, C) and Move(A, B) be executed in any order?

# Ordering Partial Solutions




**ACTIONS:**

Move(X, Y)
Precond: Clear(X), Clear(Y)
Effect: On(X, Y)

Move(X, Table)
Precond: Clear(X)
Effect: On(X, Table)

Move(A, B) removes the Clear(B) predicate which is essential for Move(B, C). Hence Move(B, C) must precede Move(A, B).

Therefore the only total order is:

1. Move(C, Table)
2. Move(B, C)
3. Move(A, B)

# Sometimes Partial Order may stay

**ACTIONS**

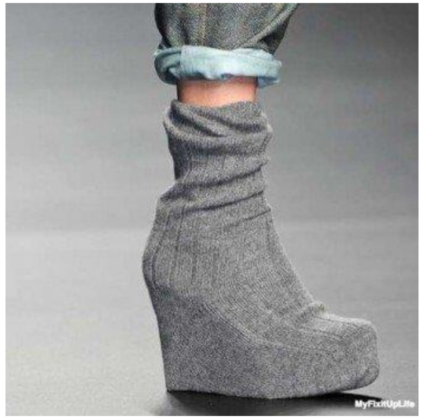
Op( **ACTION:** RightShoe,
**PRECOND:** RightSockOn,
**EFFECT:** RightShoeOn )

Op( **ACTION:** RightSock,
**EFFECT:** RightSockOn )

Op( **ACTION:** LeftShoe,
**PRECOND:** LeftSockOn,
**EFFECT:** LeftShoeOn )

Op( **ACTION:** LeftSock,
**EFFECT:** LeftSockOn )

**Which of these situations are allowed by these actions?**



# Sometimes Partial Order may stay

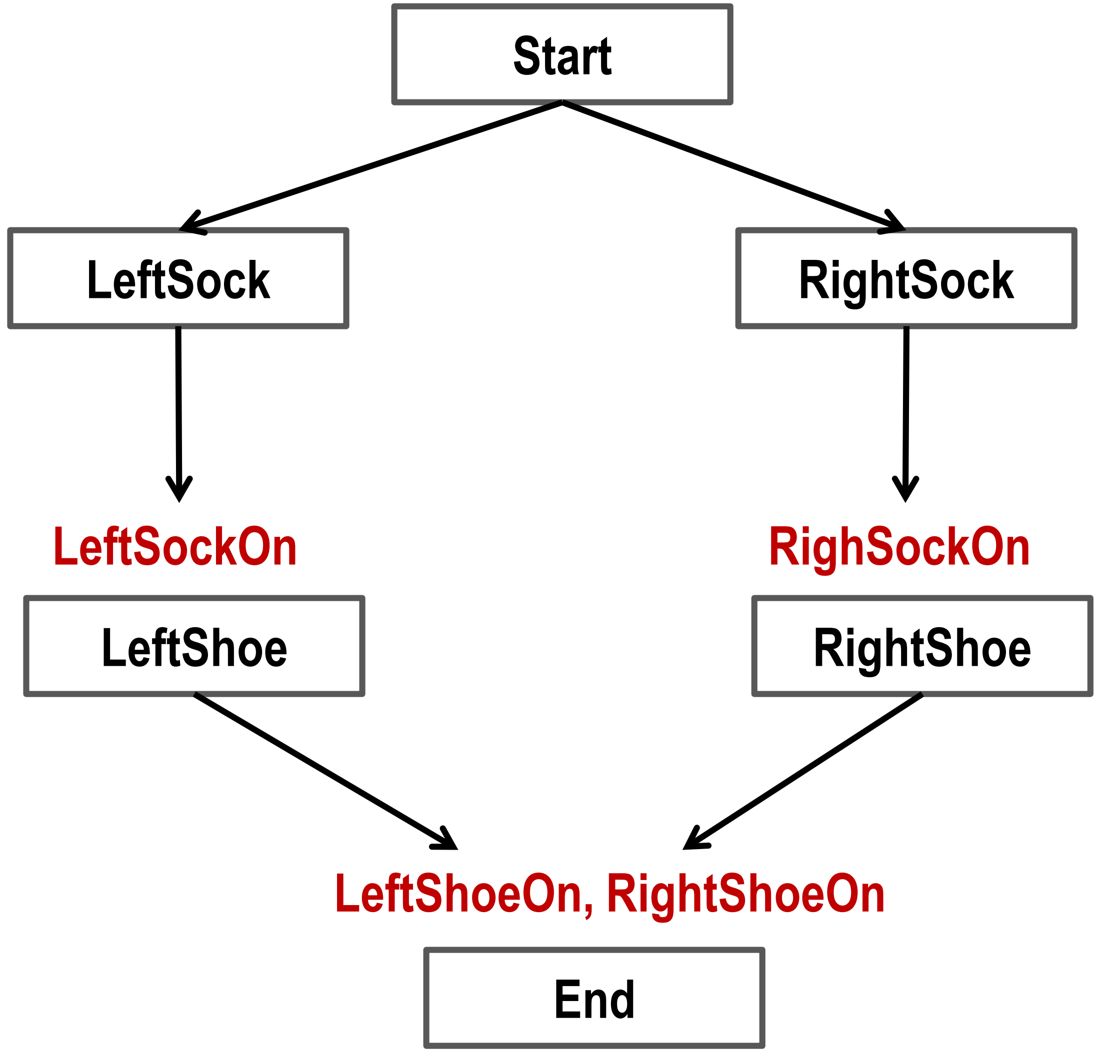
**ACTIONS**

Op( **ACTION:** RightShoe,
**PRECOND:** RightSockOn,
**EFFECT:** RightShoeOn )

Op( **ACTION:** RightSock,
**EFFECT:** RightSockOn )

Op( **ACTION:** LeftShoe,
**PRECOND:** LeftSockOn,
**EFFECT:** LeftShoeOn )

Op( **ACTION:** LeftSock,
**EFFECT:** LeftSockOn )



# Planning is an integral part of automation

**Recommended clip from Charlie Chaplin's Modern Times to see what can go wrong:**
https://www.youtube.com/watch?v=n_1apYo6-Ow

**What we intend to learn:**

1. Partial Order Planning
2. GraphPlan and SATPlan

# Partial Order Planning

- **Basic Idea:** Make choices only that are relevant to solving the current part of the problem
- **Least Commitment Choices**
  - **Orderings:** Leave actions unordered, unless they must be sequential
  - **Bindings:** Leave variables unbound, unless needed to unify with conditions being achieved
  - **Actions:** Usually not subject to “least commitment”

# Terminology

- Totally Ordered Plan
  - There exists sufficient orderings O such that all actions in A are ordered with respect to each other
- Fully Instantiated Plan
  - There exists sufficient constraints in B such that all variables are constrained to be equal to some constant
- Consistent Plan
  - There are no contradictions in O or B
- Complete Plan
  - Every precondition P of every action $A_i$ in A is achieved:
    - There exists an effect of an action $A_j$ that comes before $A_i$ and unifies with P, and no action $A_k$ that deletes P comes between $A_j$ and $A_i$

# Early Days: STRIPS

- STanford Research Institute Problem Solver
- Many planners today use specification languages that are variants of the one used in STRIPS

Our running example:

- Given:
    - Initial state: The agent is at *home* without tea, biscuits, book
    - Goal state: The agent is at *home* with tea, biscuits, book
    - A set of actions as shown next

# Representing States

- States are represented by conjunctions of function-free ground literals

  $$At(Home) \land \neg Have(Tea) \land \neg Have(Biscuits) \land \neg Have(Book)$$

- Goals are also described by conjunctions of literals

  $$At(Home) \land Have(Tea) \land Have(Biscuits) \land Have(Book)$$

- Goals can also contain variables

  $$At(x) \land Sells(x, Tea)$$

  - The above goal is *being at a shop that sells tea*

# Representing Actions

- **Action description – serves as a name**
- **Precondition – a conjunction of positive literals (why positive?)**
- **Effect – a conjunction of literals (+ve or –ve)**
    - **The original version had an *add list* and a *delete list*.**

**Op( ACTION: Go(there),**
**PRECOND: $At(here) \wedge Path(here, there)$,**
**EFFECT: $At(there) \wedge \neg At(here)$ )**

# Representing Plans

- A set of plan steps. Each step is one of the operators for the problem.
- A set of step ordering constraints. Each ordering constraint is of the form $S_i \prec S_j$, indicating $S_i$ must occur sometime before $S_j$.
- A set of variable binding constraints of the form v = x, where v is a variable in some step, and x is either a constant or another variable.
- A set of causal links written as $S \rightarrow c: S'$ indicating S satisfies the precondition c for S'.

# Example

- **Initial plan**

Plan(
STEPS: {
S1: Op( ACTION: start ),
S2: Op( ACTION: finish,
PRECOND: RightShoeOn $\wedge$ LeftShoeOn )
},
ORDERINGS: $\{S_1 \prec S_2\}$,
BINDINGS: { },
LINKS: { } )

# POP Example: Get Tea, Biscuits, Book

**Initial state:**

Op( **ACTION:** Start,
  **EFFECT:** At(Home) $\wedge$ Sells(BS, Book)
    $\wedge$ Sells(TS, Tea)
    $\wedge$ Sells(TS, Biscuits) )

**Goal state:**

Op( **ACTION:** Finish,
  **PRECOND:** At(Home) $\wedge$ Have(Tea)
    $\wedge$ Have(Biscuits)
    $\wedge$ Have(Book) )

**Actions:**

Op( **ACTION:** Go(y),
  **PRECOND:** At(x),
  **EFFECT:** At(y) $\wedge$ $\neg$At(x))

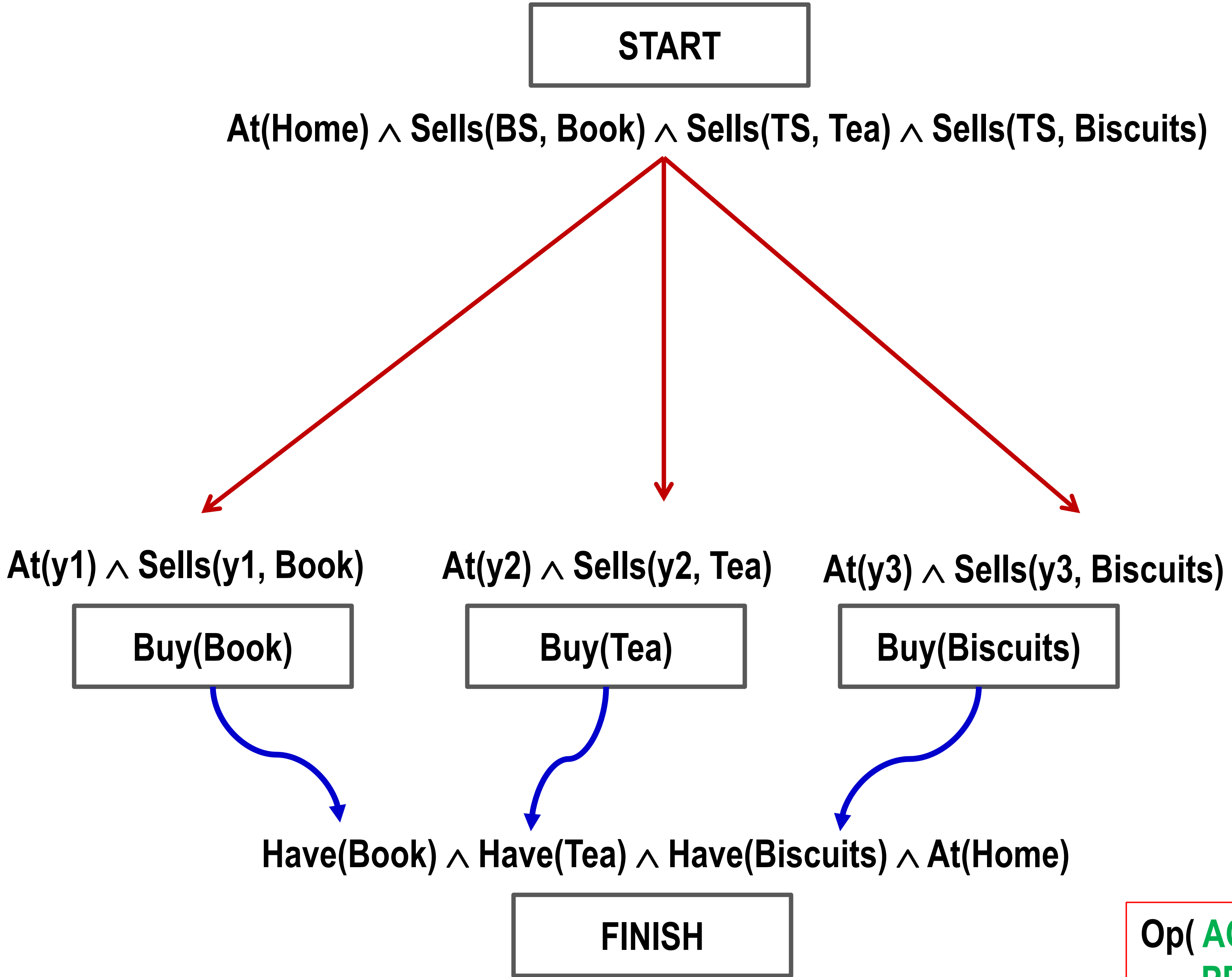
Op( **ACTION:** Buy(x),
  **PRECOND:** At(y) $\wedge$ Sells(y, x),
  **EFFECT:** Have(x))

**START**

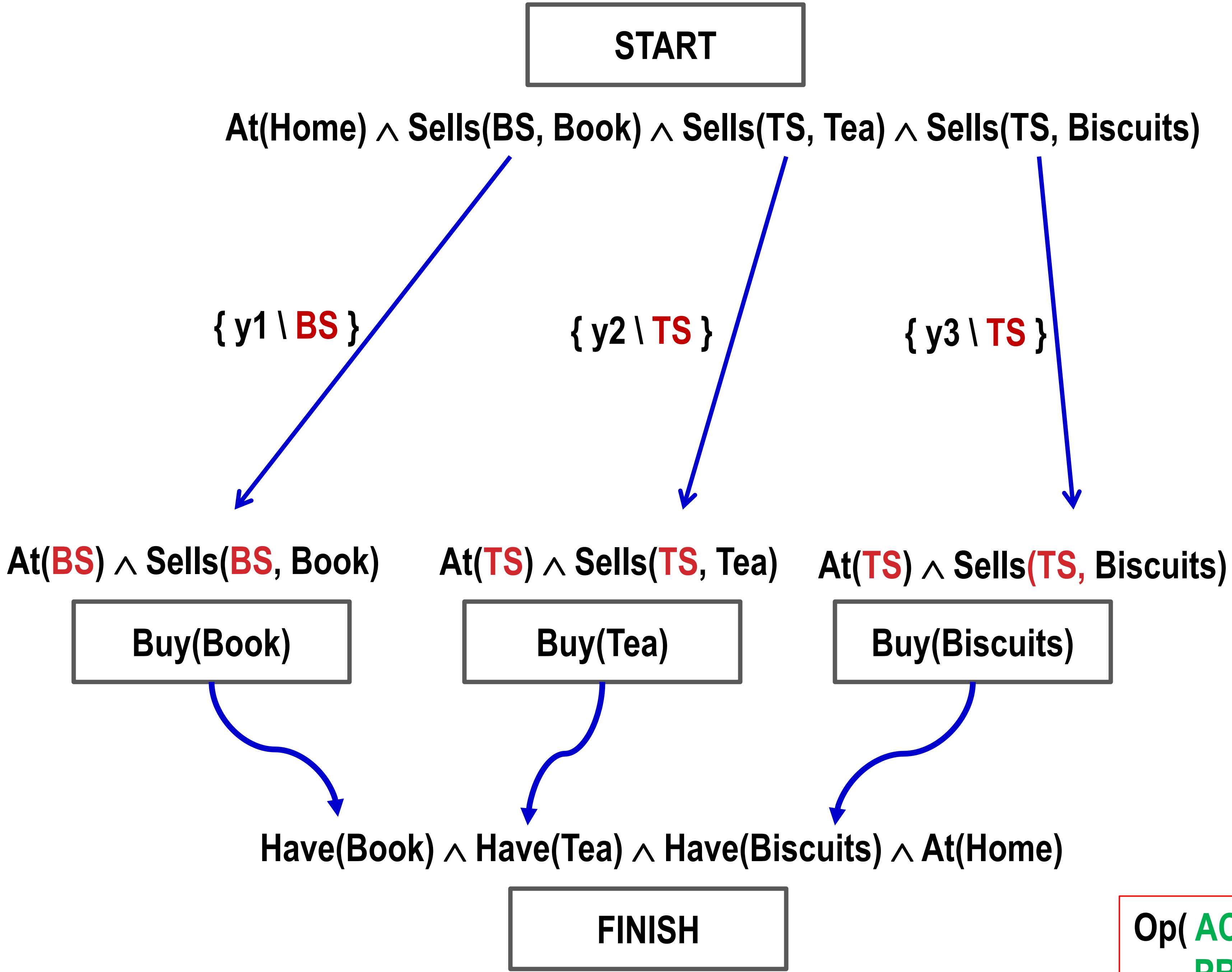
At(Home) $\wedge$ Sells(BS, Book) $\wedge$ Sells(TS, Tea) $\wedge$ Sells(TS, Biscuits)

Have(Book) $\wedge$ Have(Tea) $\wedge$ Have(Biscuits) $\wedge$ At(Home)

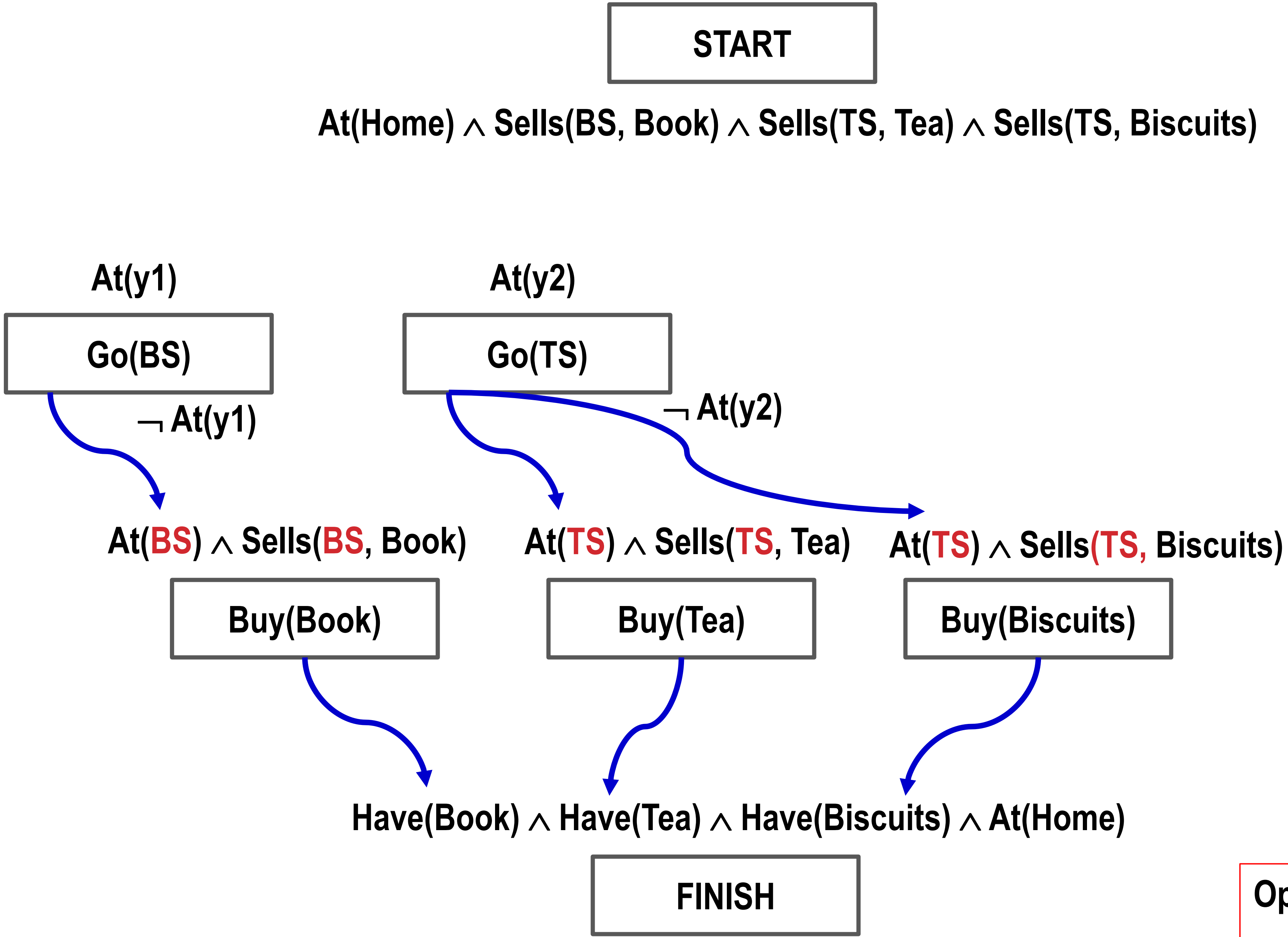
**FINISH**



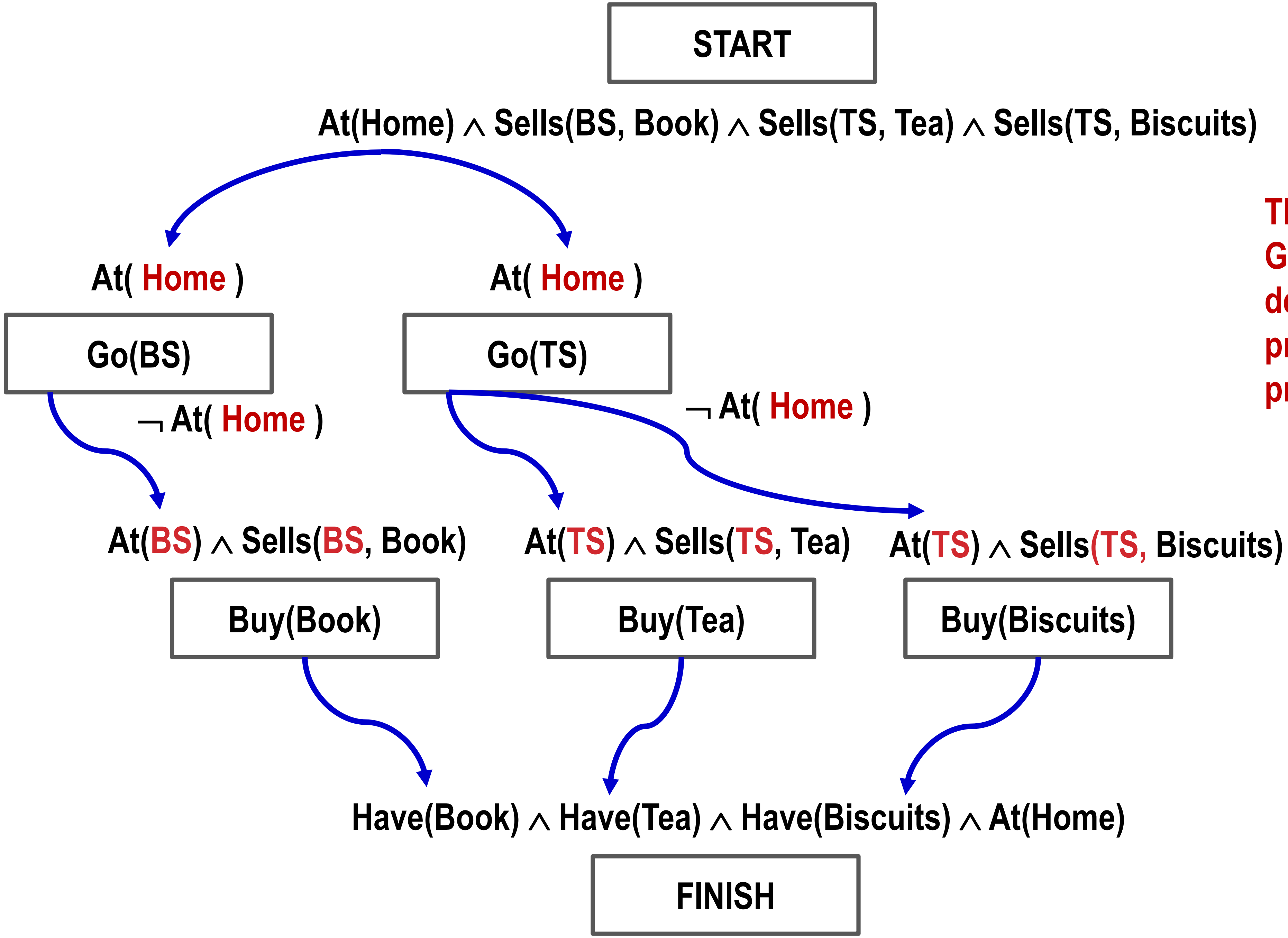

Op( ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))



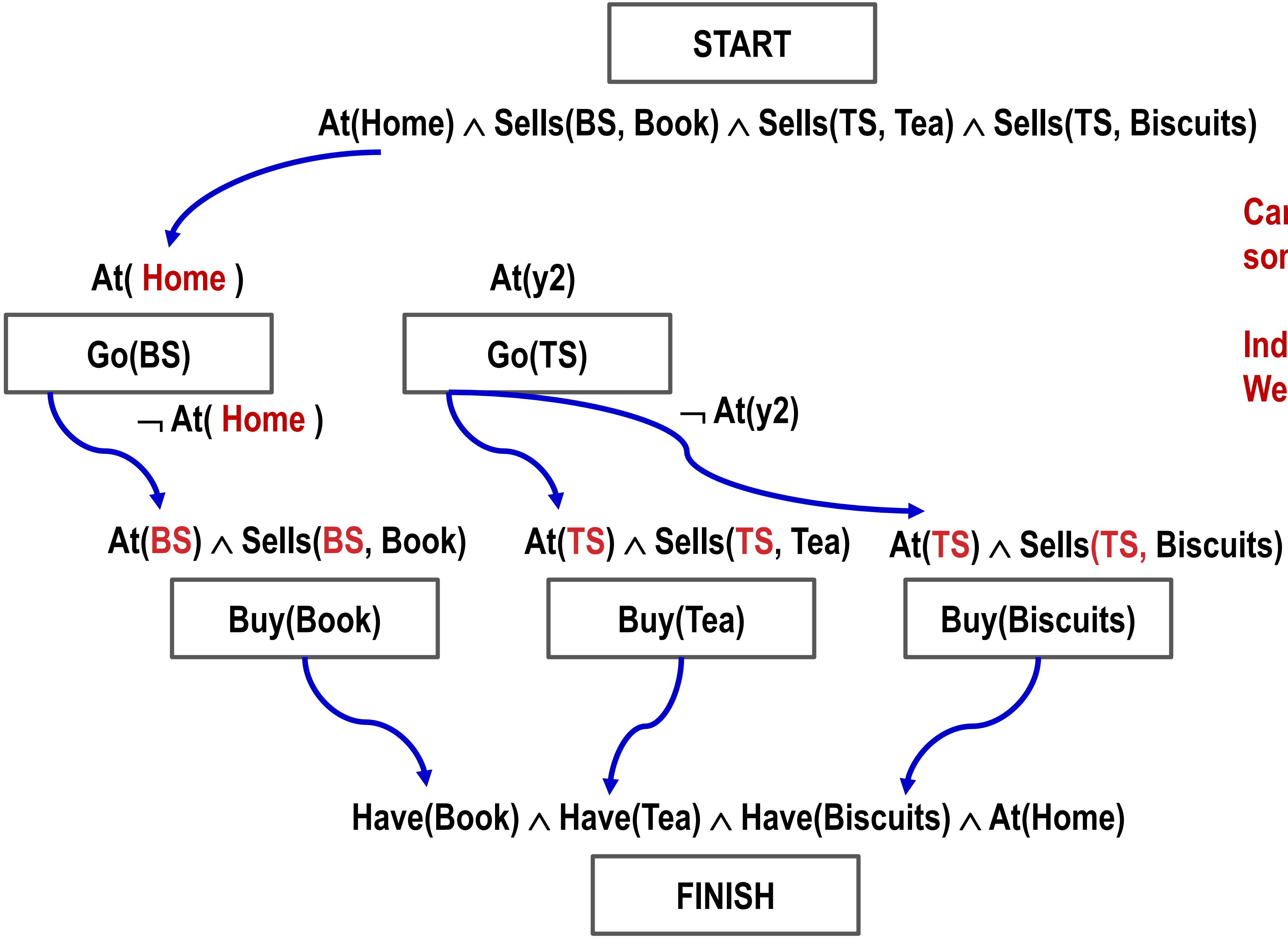

Op( ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))




Op( ACTION: Go(y),
PRECOND: At(x),
EFFECT: At(y) ∧ ¬At(x))

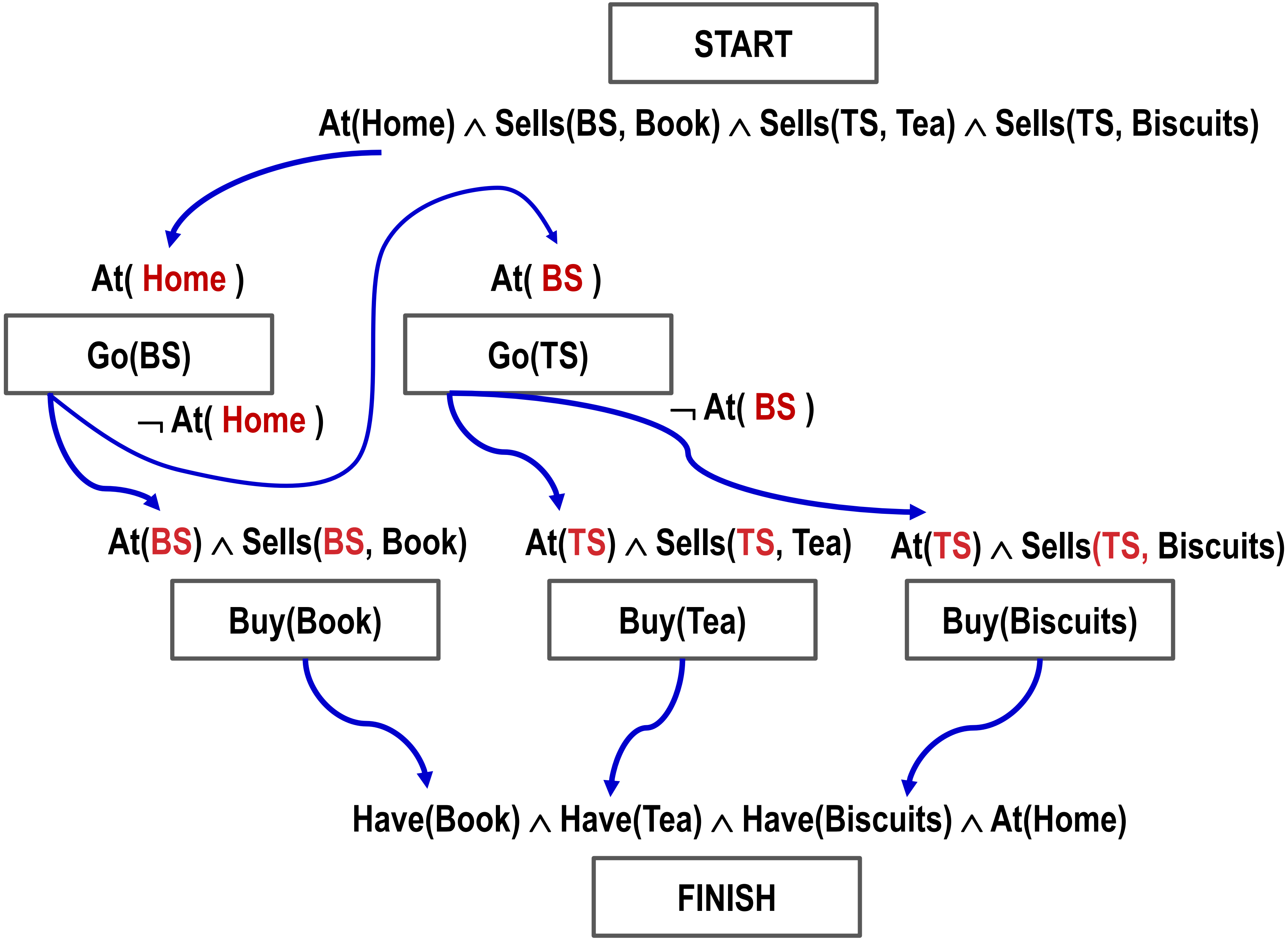


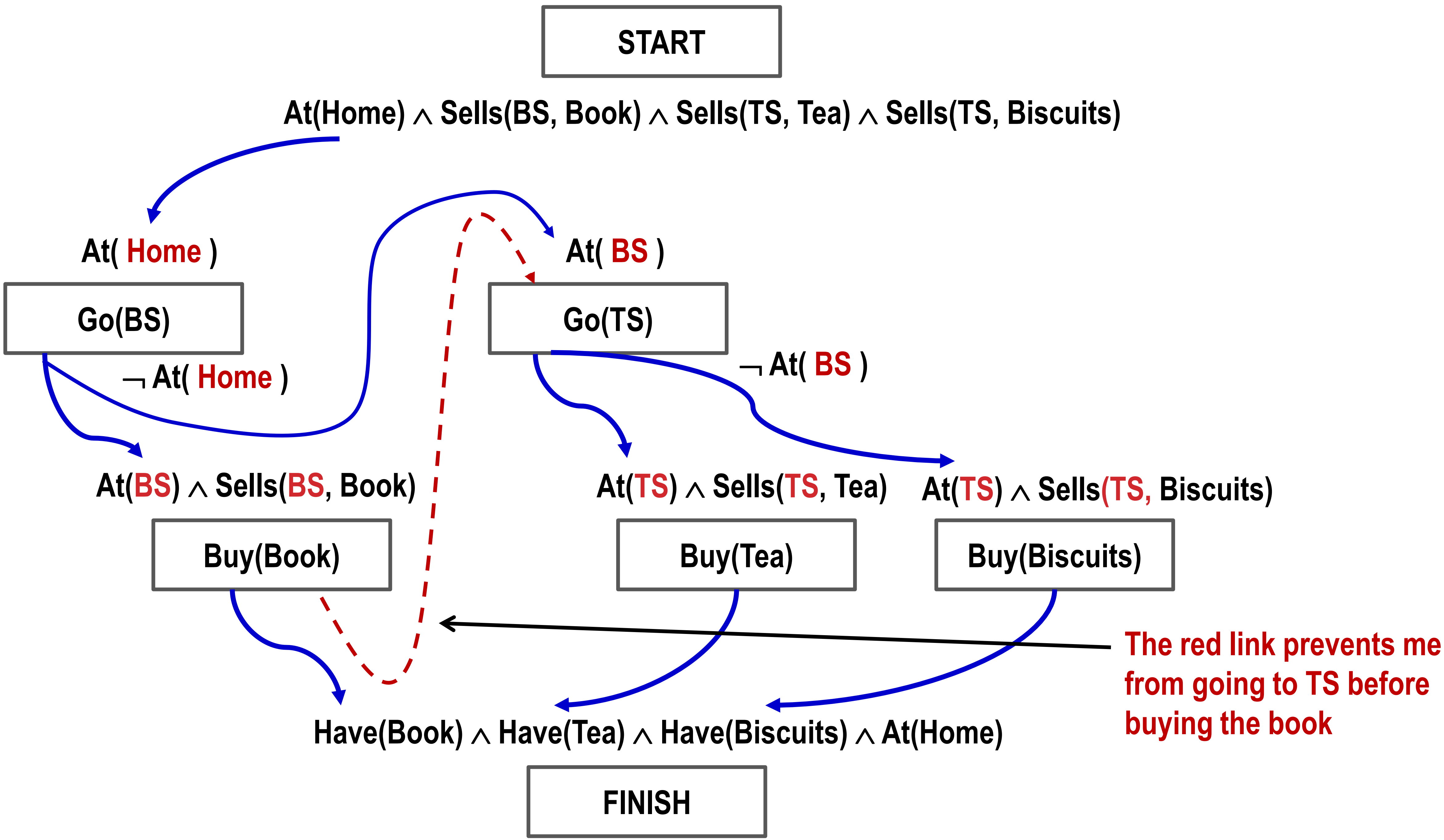

The problem here is that Go(BS) and Go(TS) destroy each other's precondition. Neither can precede the other.

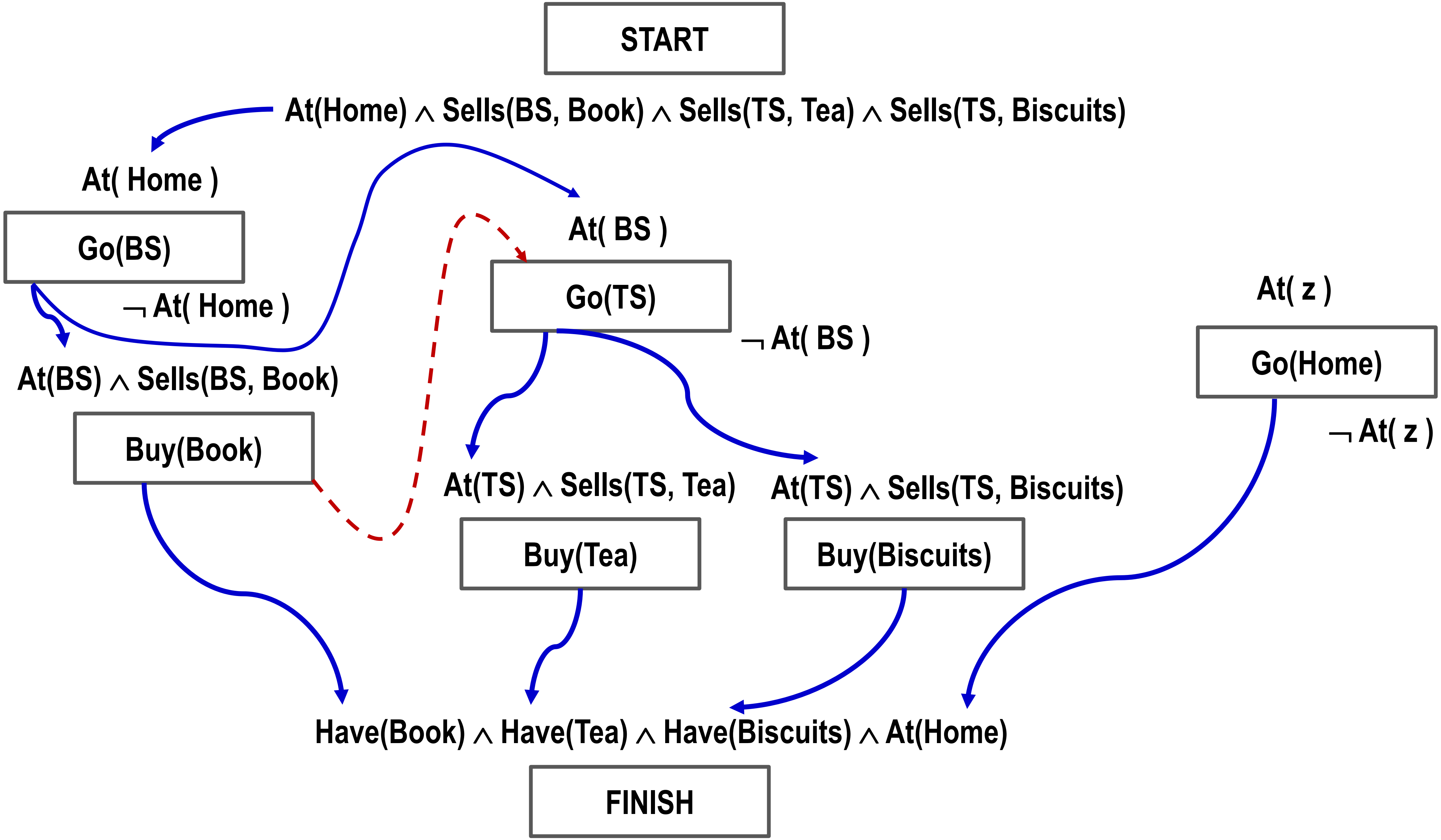


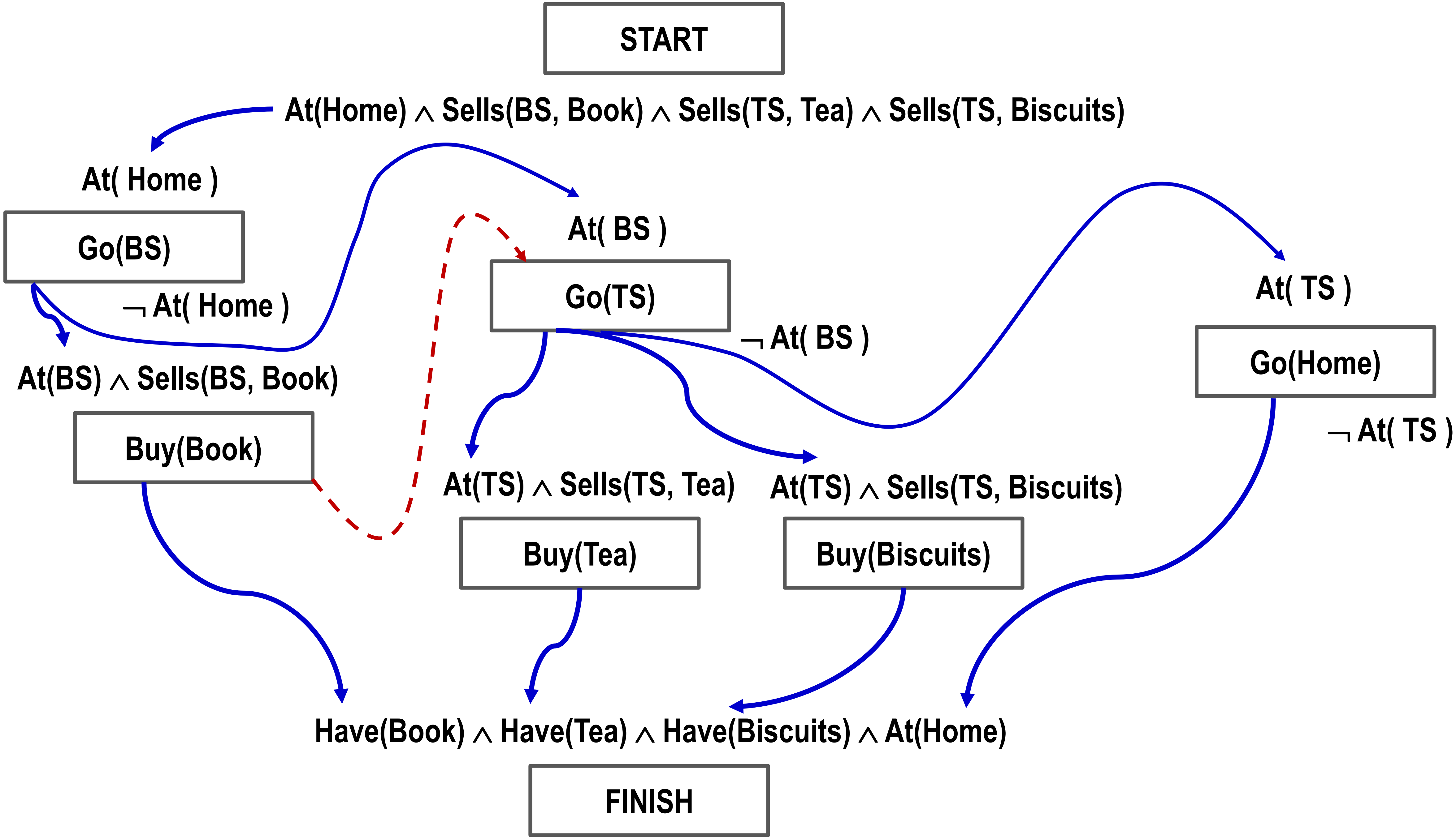

Can y2 be instantiated with something else?

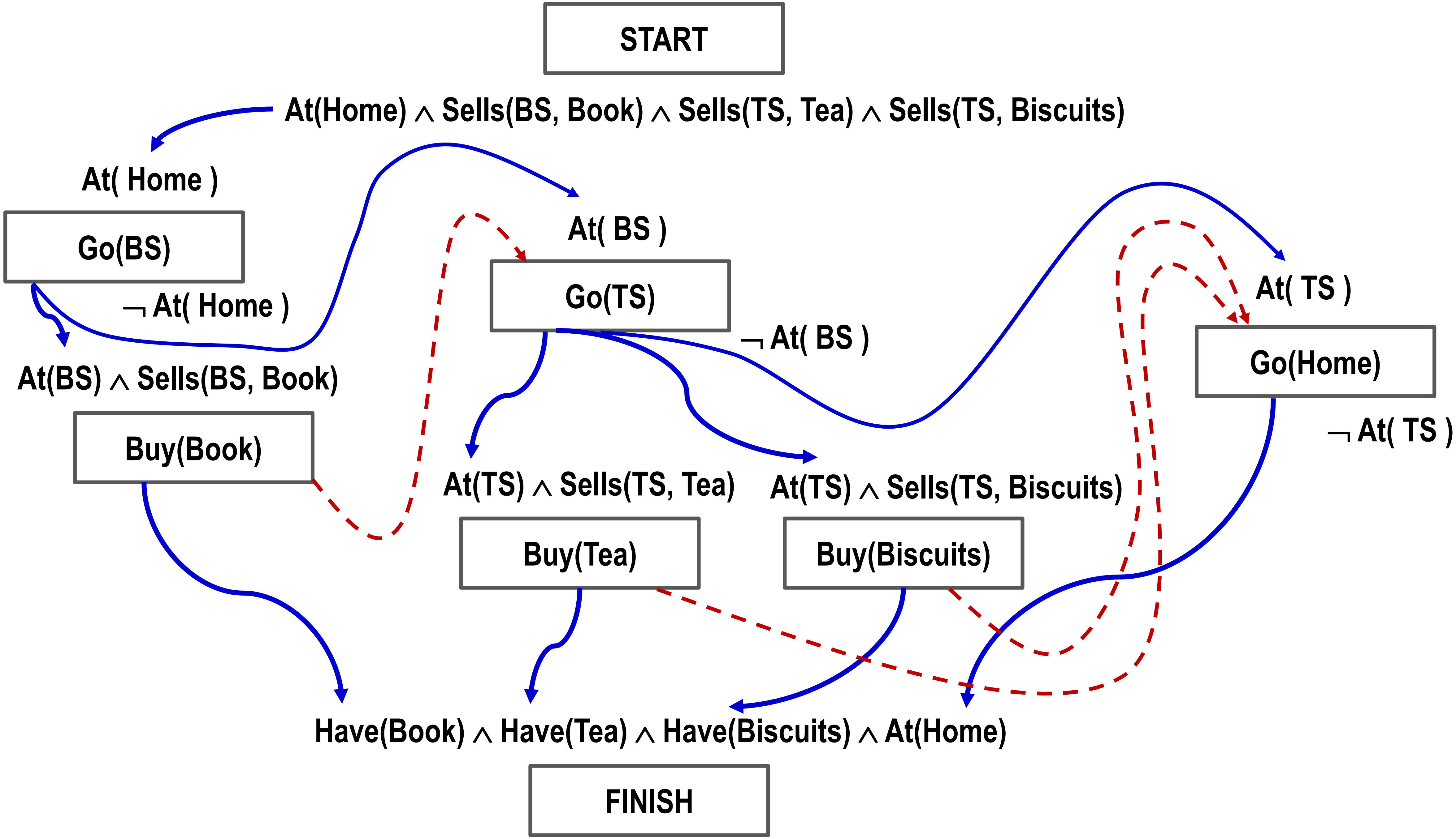
Indeed !!
We can try BS for example.

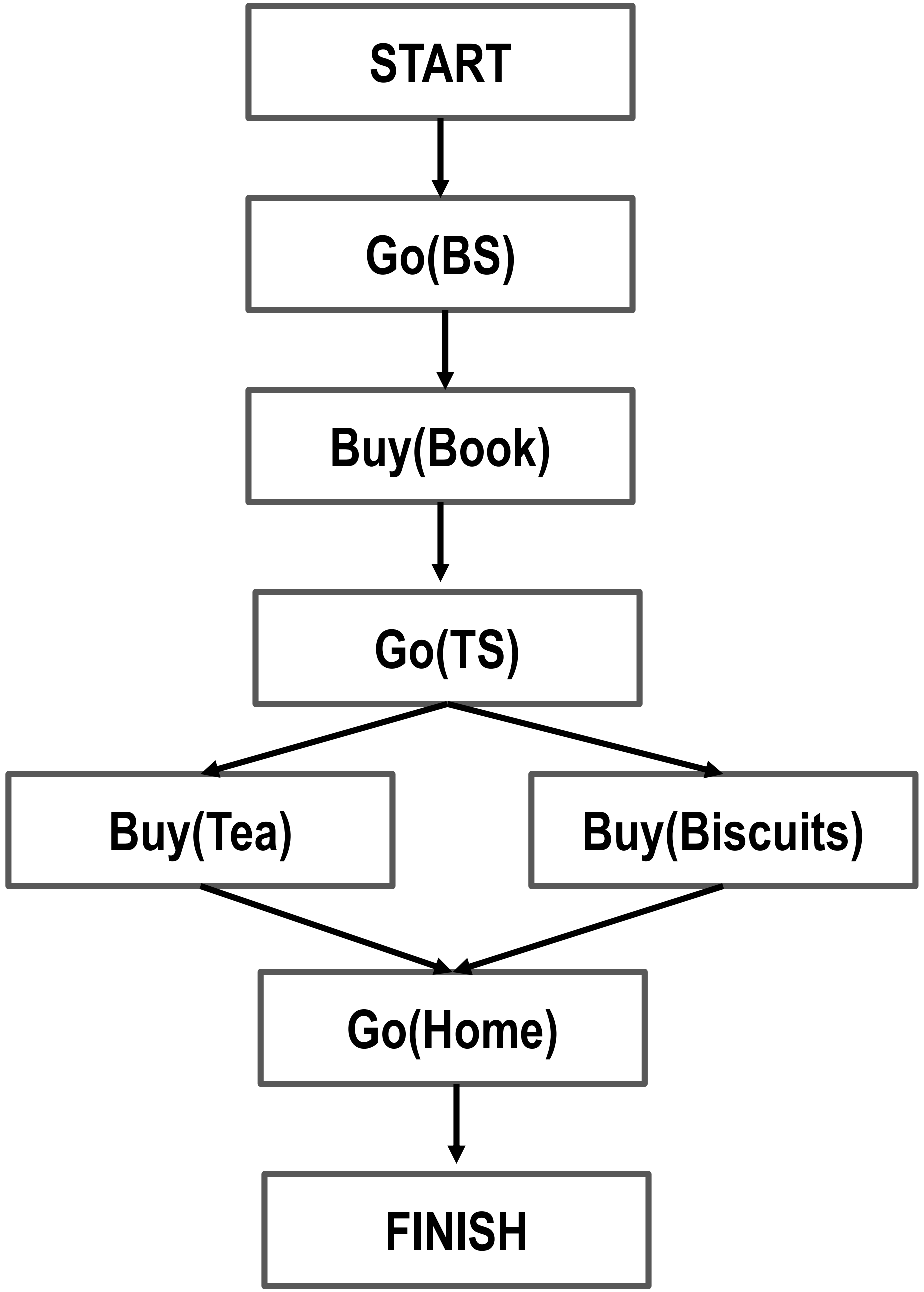

START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At( Home )
At( BS )
Go(BS)
Go(TS)
¬ At( Home )
¬ At( BS )
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At( Home )
At( BS )
Go(BS)
Go(TS)
¬ At( Home )
¬ At( BS )
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
The red link prevents me from going to TS before buying the book
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At( Home )
Go(BS)
¬ At( Home )
At(BS) ∧ Sells(BS, Book)
Buy(Book)
At( BS )
Go(TS)
¬ At( BS )
At(TS) ∧ Sells(TS, Tea)
Buy(Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Biscuits)
At( z )
Go(Home)
¬ At( z )
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At( Home )
Go(BS)
¬ At( Home )
At(BS) ∧ Sells(BS, Book)
Buy(Book)
At( BS )
Go(TS)
¬ At( BS )
At(TS) ∧ Sells(TS, Tea)
Buy(Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Biscuits)
At( TS )
Go(Home)
¬ At( TS )
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At( Home )
Go(BS)
¬ At( Home )
At(BS) ∧ Sells(BS, Book)
Buy(Book)
At( BS )
Go(TS)
¬ At( BS )
At(TS) ∧ Sells(TS, Tea)
Buy(Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Biscuits)
At( TS )
Go(Home)
¬ At( TS )
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


START
Go(BS)
Buy(Book)
Go(TS)
Buy(Tea)
Buy(Biscuits)
Go(Home)
FINISH

# The Partial Order Planning Algorithm

```
Function POP( initial, goal, operators )
// Returns plan
        plan ← Make-Minimal-Plan( initial, goal )
        Loop do
                If Solution( plan ) then return plan
                S, c ← Select-Subgoal( plan )
                Choose-Operator( plan, operators, S, c )
                Resolve-Threats( plan )
        end
```

# POP: Selecting Sub-Goals

Function Select-Subgoal( *plan* )

// Returns S, c

pick a plan step S from STEPS( *plan* )

with a precondition c that has not been achieved

Return S, c

# POP: Choosing operators

**Procedure Choose-Operator( *plan, operators*, S, c )**

Choose a step S' from *operators* or STEPS( *plan* ) that has c as an effect

If there is no such step then fail

Add the causal link S' $\rightarrow$ c: S to LINKS( *plan* )

Add the ordering constraint S' $\prec$ S to ORDERINGS( *plan* )

If S' is a newly added step from *operators* then add S' to STEPS( *plan* ) and add Start $\prec$ S' $\prec$ Finish to ORDERINGS( *plan* )

# POP: Resolving Threats

Procedure Resolve-Threats( *plan* )

for each S' that threatens a link $S_i \rightarrow c: S_j$ in LINKS( *plan* ) do

choose either

*Promotion:* Add $S' \prec S_i$ to ORDERINGS( *plan* )

*Demotion:* Add $S_j \prec S'$ to ORDERINGS( *plan* )

if not Consistent( *plan* ) then fail

# Partially instantiated operators

- So far we have not mentioned anything about binding constraints
- Should an operator that has the effect, say, $\neg$*At(x)*, be considered a threat to the condition, *At(Home)* ?
    - Indeed it is a *possible threat* because *x* may be bound to *Home*

# Dealing with potential threats

- Resolve now with an equality constraint
  - Bind x to something that resolves the threat (say *x* = *TS*)
- Resolve now with an inequality constraint
  - Extend the language of variable binding to allow $x \neq Home$
- Resolve later
  - Ignore possible threats. If *x* = *Home* is added later into the plan, then we will attempt to resolve the threat (by promotion or demotion)

**Proc Choose-Operator( *plan, operators*, S, c )**

choose a step S' from *operators* or STEPS( *plan* ) that has c' as an effect

such that *u* = UNIFY( c, c', BINDINGS( plan ))

if there is no such step then fail

add *u* to BINDINGS( *plan* )

add the causal link S' $\rightarrow$ c: S to LINKS( *plan* )

add the ordering constraint S' $\prec$ S to ORDERINGS( *plan* )

if S' is a newly added step from *operators* then

add S' to STEPS( *plan* ) and add Start $\prec$ S' $\prec$ Finish to ORDERINGS( *plan* )

**Procedure Resolve-Threats( *plan* )**

```
for each Si → c: Sj in LINKS( plan ) do
  for each S" in STEPS( plan ) do
      for each c' in EFFECTS( S") do
        if SUBST( BINDINGS(plan), c ) = SUBST( BINDINGS(plan), ¬c' )
        then choose either
            Promotion: Add S" ≺ Si to ORDERINGS( plan )
            Demotion: Add Sj ≺ S" to ORDERINGS( plan )
        if not Consistent( plan ) then fail
```

## USING PLANNING GRAPHS

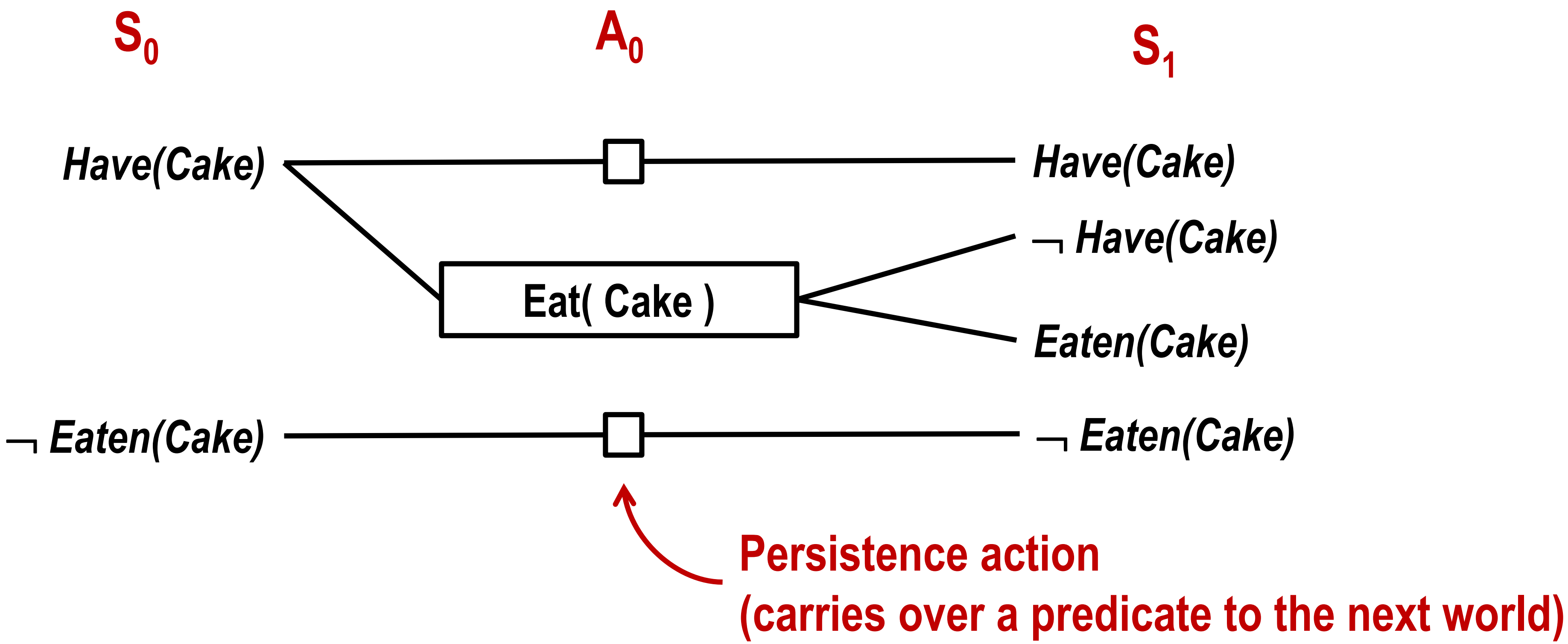
# GraphPlan and SATPlan

# Planning Graph

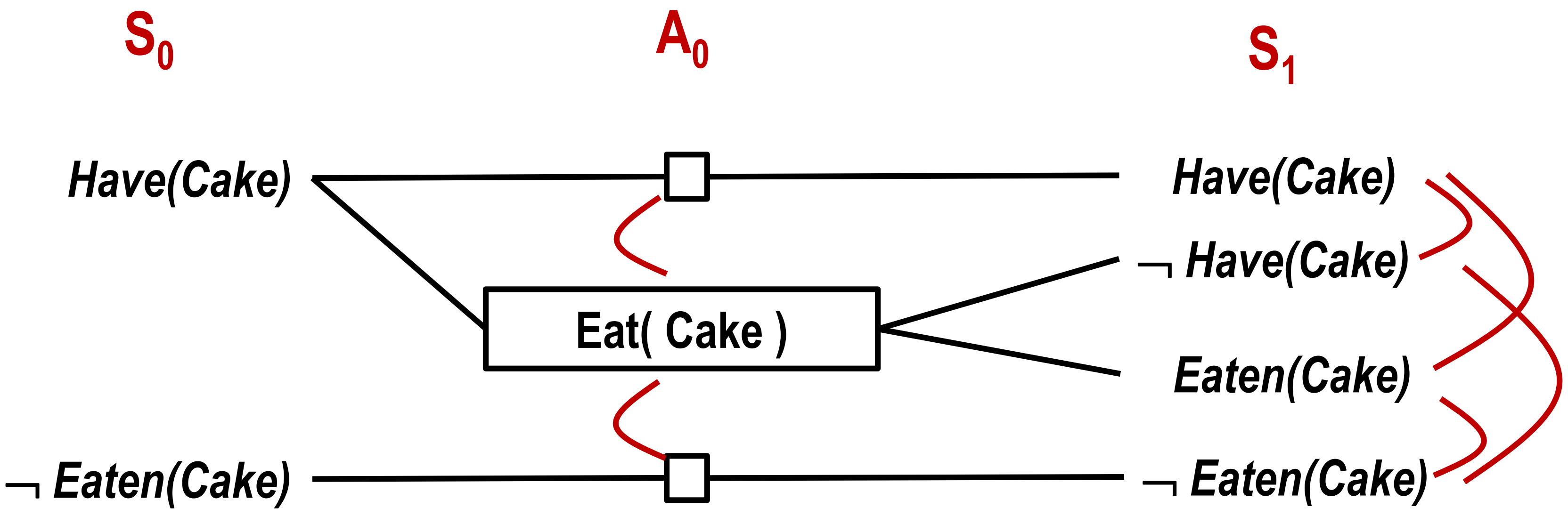
Op( **ACTION**: Eat(Cake),
**PRECOND**: Have(Cake),
**EFFECT**: Eaten(Cake) ∧ ¬Have(Cake))

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

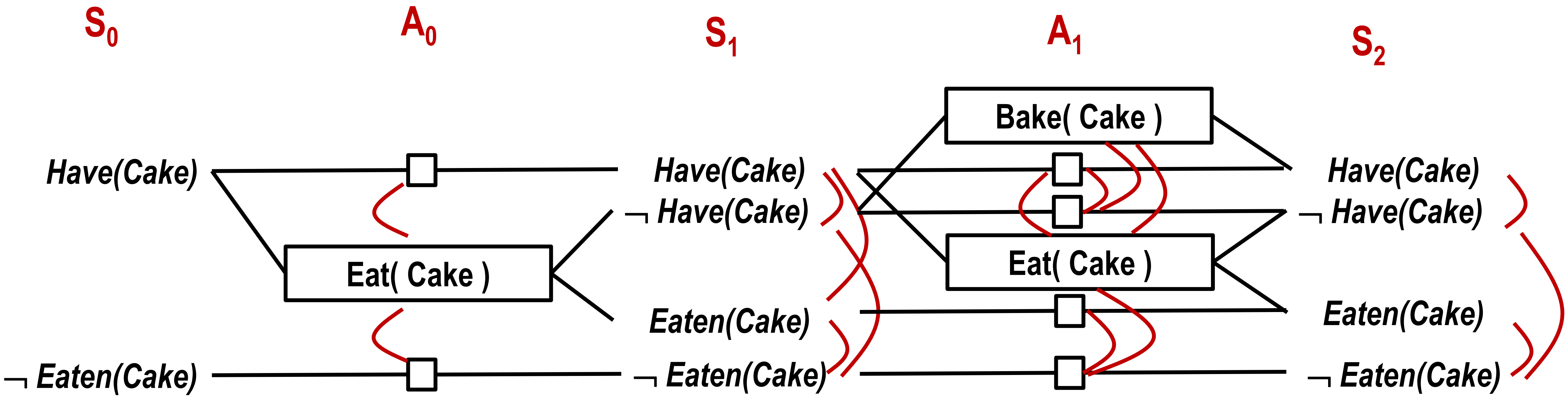
# Planning Graphs

- 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) ∧ ¬ Eaten(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

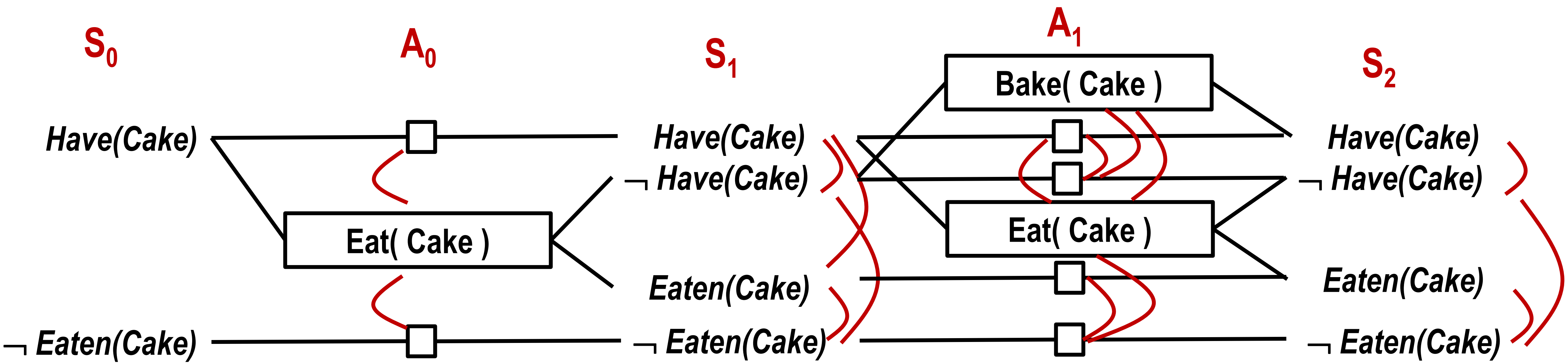
# Mutex Actions

- **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**

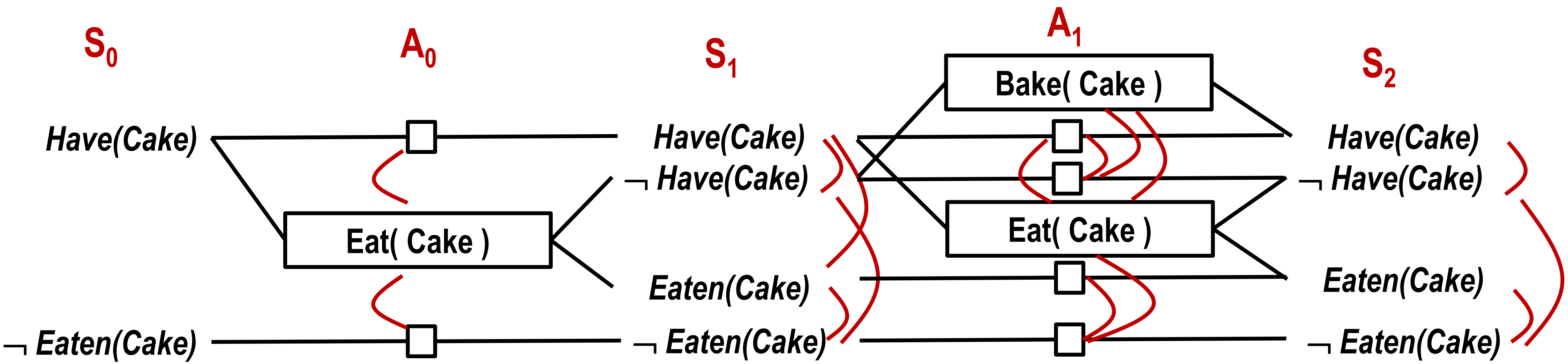
    **Eat( Cake ) causes ¬ *Have(Cake)* and the persistence of *Have( Cake )* needs *Have(Cake)***
  - **Competing needs – one of the preconditions of one action is mutually exclusive with a precondition of the other**

    **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)

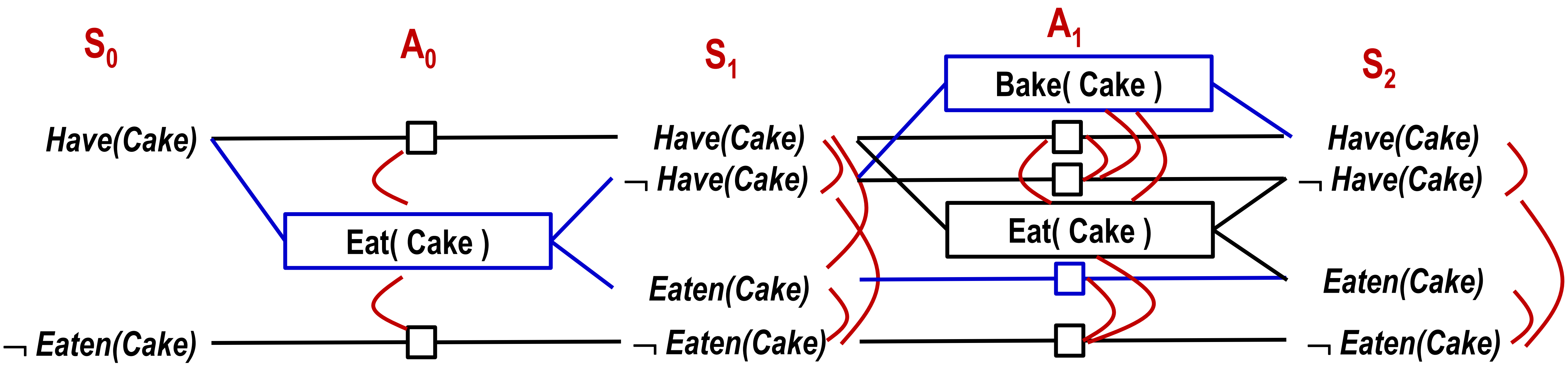


## Function GraphPLAN( problem )

```
// 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

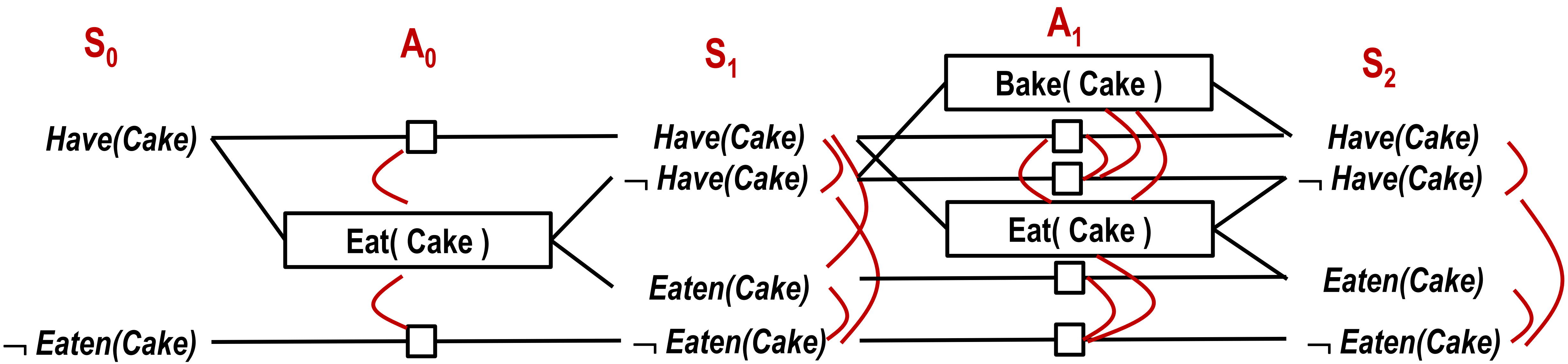
- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness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

Goal: At( Spare, Axle )

Op( ACTION: Remove( Spare, Trunk ),
PRECOND: At( Spare, Trunk ),
EFFECT: At( Spare, Ground )
$\wedge \neg$ At( Spare, Trunk ))

Op( ACTION: Remove( Flat, Axle ),
PRECOND: At( Flat, Axle ),
EFFECT: At( Flat, Ground )
$\wedge \neg$ At( Flat, Axle ))

Op( ACTION: PutOn( Spare, Axle ),
PRECOND: At( Spare, Ground )
$\wedge \neg$ At( Flat, Axle ),
EFFECT: At( Spare, Axle )
$\wedge \neg$ At( Spare, Ground ))

Op( ACTION: LeaveOvernight,
PRECOND:
EFFECT: $\neg$ At( Spare, Ground )
$\wedge \neg$ At( Spare, Axle )
$\wedge \neg$ At( Spare, Trunk )
$\wedge \neg$ At( Flat, Ground )
$\wedge \neg$ At( Flat, Axle ))

# 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:**

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

**State Constraints:**

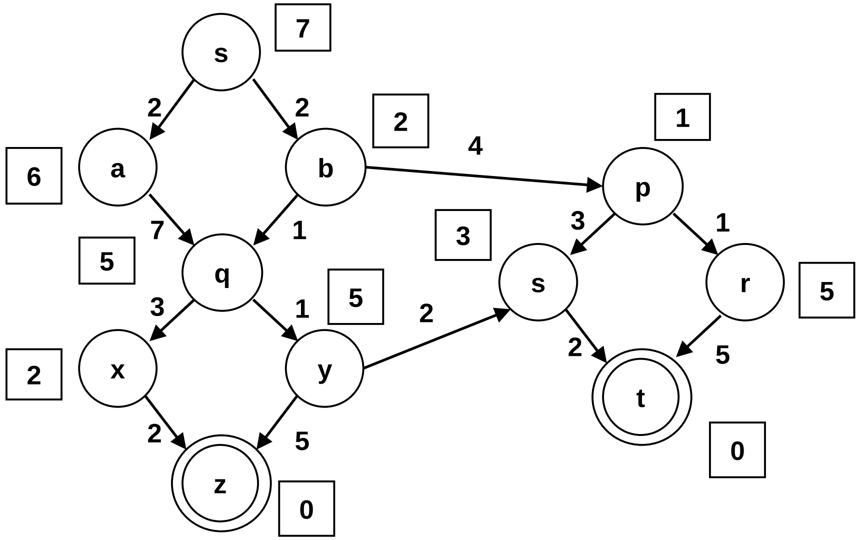
$$\forall p, x, y, t \;\; (x \neq y) \Rightarrow \neg (At(p, x)^t \land At(p, y)^t)$$

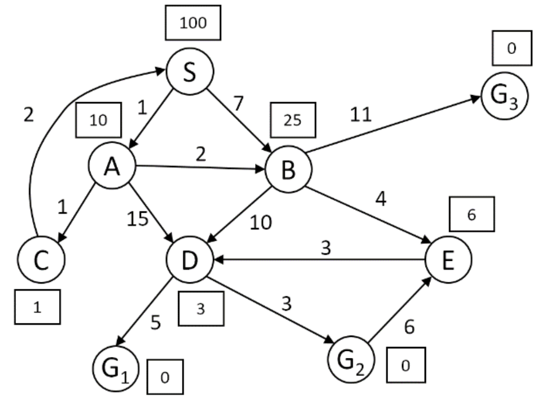
# SATPlan

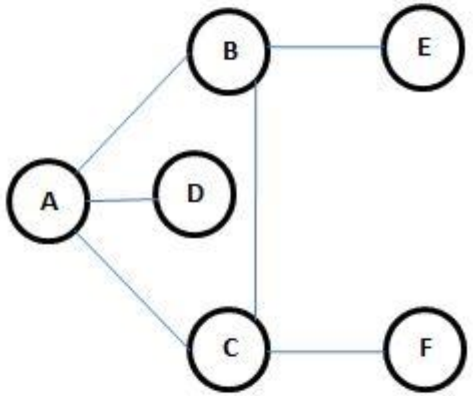
**Function SATPlan( problem, $T_{max}$ )**

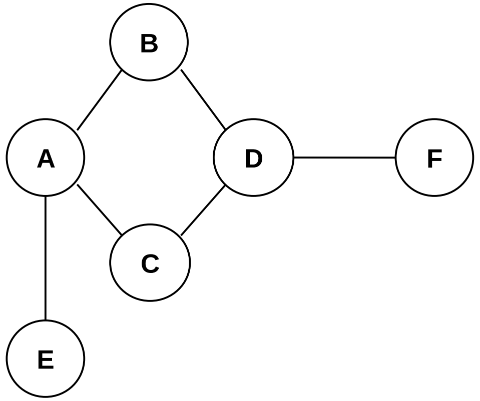
// *returns solution or failure*

```
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
```


s
7
2
2
a
6
b
2
4
p
1
7
1
q
5
3
1
5
3
3
1
s
r
5
2
x
2
y
2
2
5
t
0
2
5
z
0


100
S
0
$G_3$
2
1
7
10
25
11
A
2
B
1
4
6
15
10
C
D
3
E
1
3
3
5
6
$G_1$
0
$G_2$
0


A
B
C
D
E
F


A
B
C
D
E
F