Planning in Artificial Intelligence

The intelligent way to do things

COURSE: CS60045

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Exercise

Consider the problem of swapping the contents of two registers, A and B. For a programmer, this is very easy, but suppose we wish to ask a robot to figure out how to write such a code. Suppose we pose it as the following planning problem in STRIPS:

Op(ACTION: Start,

EFFECT: Contains(A, X) $\wedge$ Contains(B, Y))

// Register A contains X, Register B contains Y

Op(ACTION: Finish,

PRECOND: Contains(B, X) $\wedge$ Contains(A, Y))

// The following action assigns the content v1 of register r1 to register r2 which contained v2

Op(ACTION: Assign(r1, v1, r2, v2),

PRECOND: Contains(r1, v1) $\wedge$ Contains(r2, v2),

EFFECT: Contains(r2, v1))

Exercise

Consider the problem of swapping the contents of two registers, A and B. For a programmer, this is very easy, but suppose we wish to ask a robot to figure out how to write such a code. Suppose we pose it as the following planning problem in STRIPS:

Op(ACTION: Start,

EFFECT: Contains(A, X) $\wedge$ Contains(B, Y))

// Register A contains X, Register B contains Y

Op(ACTION: Finish,

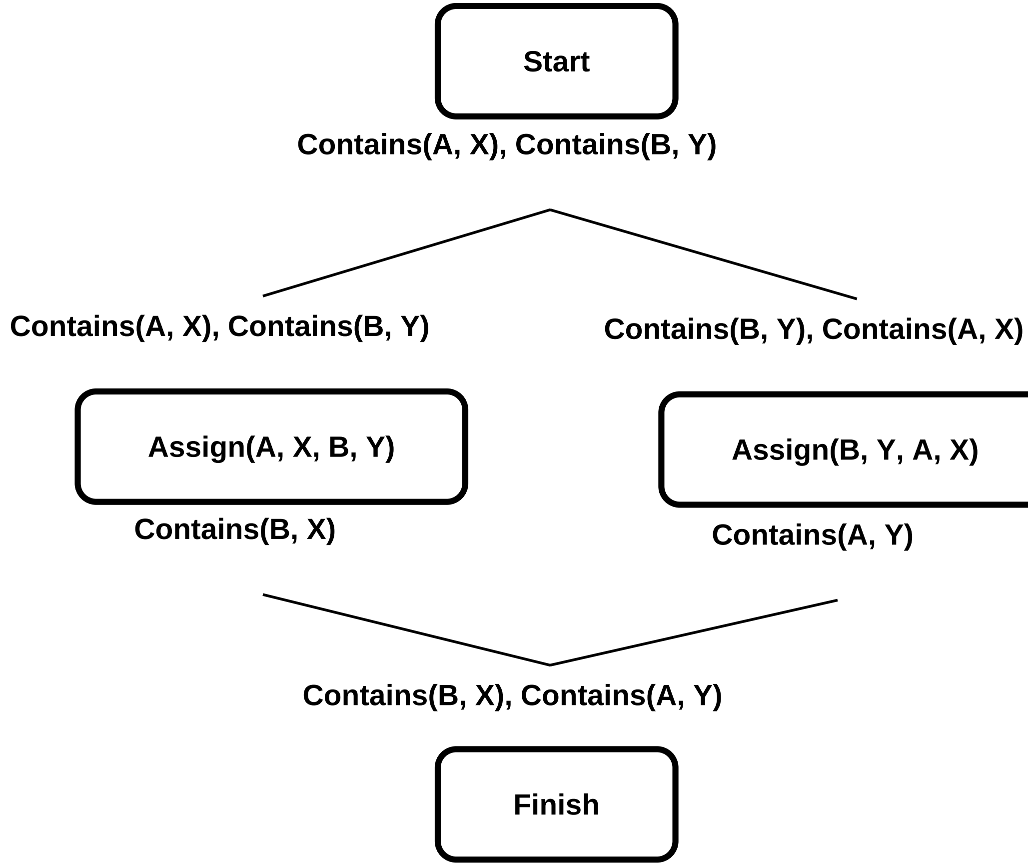
PRECOND: Contains(B, X) $\wedge$ Contains(A, Y))

// The following action assigns the content v1 of register r1 to register r2 which contained v2

Op(ACTION: Assign(r1, v1, r2, v2),

PRECOND: Contains(r1, v1) $\wedge$ Contains(r2, v2),

EFFECT: Contains(r2, v1))



Observe that the steps of the plan cannot be executed in any order to achieve the swapping the contents of the registers. The robot is not at fault, since it was not told that assigning the contents of register r1 to register r2 destroys the previous content of register r2. Can you rewrite the action so that the correct consequence of the action is captured?

Exercise

Consider the problem of swapping the contents of two registers, A and B. For a programmer, this is very easy, but suppose we wish to ask a robot to figure out how to write such a code. Suppose we pose it as the following planning problem in STRIPS:

Op(ACTION: Start,

EFFECT: Contains(A, X) ∧ Contains(B, Y))

// Register A contains X, Register B contains Y

Op(ACTION: Finish,

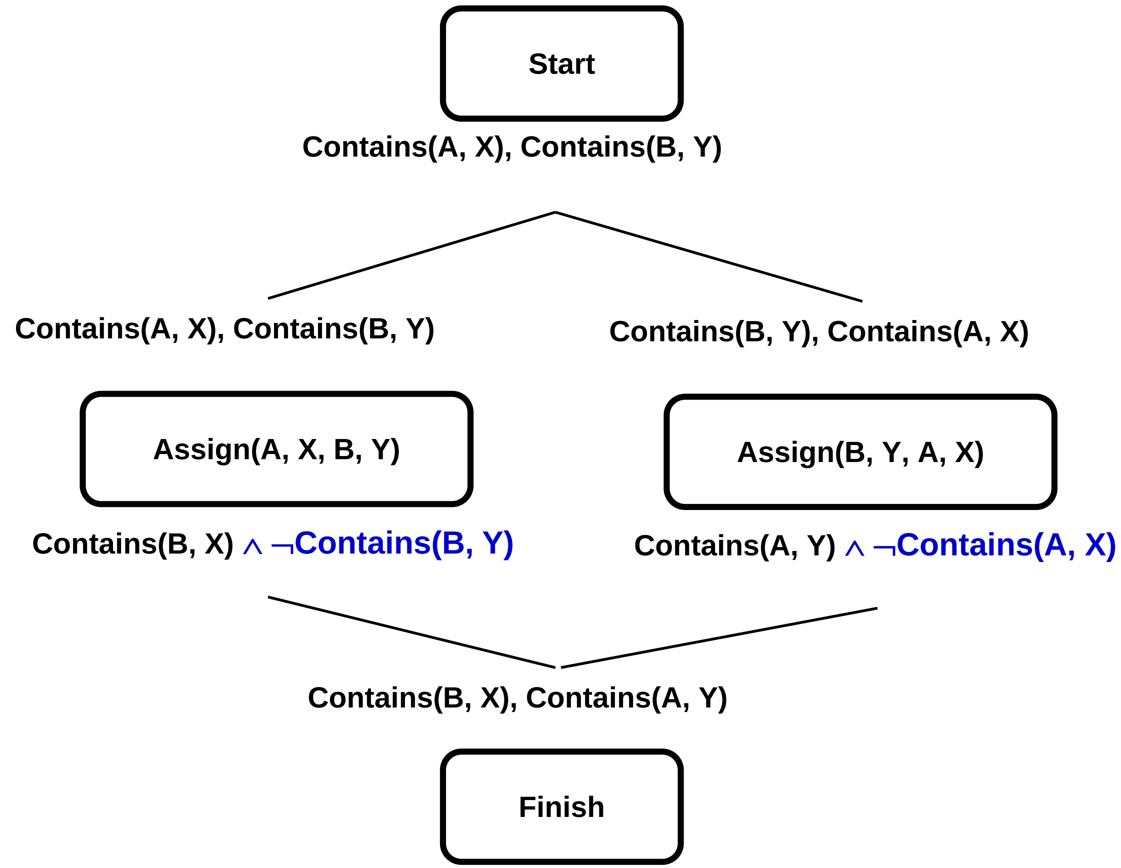
PRECOND: Contains(B, X) ∧ Contains(A, Y))

// The following action assigns the content v1 of register r1 to register r2 which contained v2

Op(ACTION: Assign(r1, v1, r2, v2),

PRECOND: Contains(r1, v1) ∧ Contains(r2, v2),

EFFECT: Contains(r2, v1) ∧ ¬Contains(r2, v2))



And now there is no order in which the steps can be executed due to a cyclic ordering constraint.

Exercise

Consider the problem of swapping the contents of two registers, A and B. For a programmer, this is very easy, but suppose we wish to ask a robot to figure out how to write such a code. Suppose we pose it as the following planning problem in STRIPS:

Op(ACTION: Start,

EFFECT: Contains(A, X) ∧ Contains(B, Y))

// Register A contains X, Register B contains Y

Op(ACTION: Finish,

PRECOND: Contains(B, X) ∧ Contains(A, Y))

// The following action assigns the content v1 of register r1 to register r2 which contained v2

Op(ACTION: Assign(r1, v1, r2, v2),

PRECOND: Contains(r1, v1) ∧ Contains(r2, v2),

EFFECT: Contains(r2, v1) ∧ ¬Contains(r2, v2))

After the modification shown in blue, observe that no totally ordered plan exists corresponding to the plan shown.

Now suppose we have a third register, C. Draw a partial order plan for swapping A and B using C and show that it can then be totally ordered.

Start
Contains(A,X) ∧ Contains(B, Y)
|
Assign(A, X, C, v2)
Contains(C, X) ∧ ¬ Contains(C, v2)
|
Assign(B, Y, A, X)
Contains(A, Y) ∧ ¬ Contains(A, X)
|
Assign(C, X, B, Y)
Contains(B, X) ∧ ¬ Contains(B, Y)
|
Finish

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”

Example

- Initial plan

```
Plan(
    STEPS: {
        S1:  Op(  ACTION: Start,
                  EFFECT: At(Home) ∧ Sells(BS, Book)
                          ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)   ),

        S2:  Op(  ACTION: Finish,
                  PRECOND: At(Home) ∧ Have(Tea)
                          ∧ Have(Biscuits) ∧ Have(Book) ),
    },
    ORDERINGS: {S1 ≺ S2},
    BINDINGS: { },
    LINKS: { }          )
```

S_1: START

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

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))

ORDERINGS: $\{S_1 \prec S_2\}$

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

S_2: 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

S_1: START

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

ORDERINGS: $\{S_1 \prec S_2\}$

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

S_2: FINISH

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*)

START

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

$At(y1) \land Sells(y1, Book)$

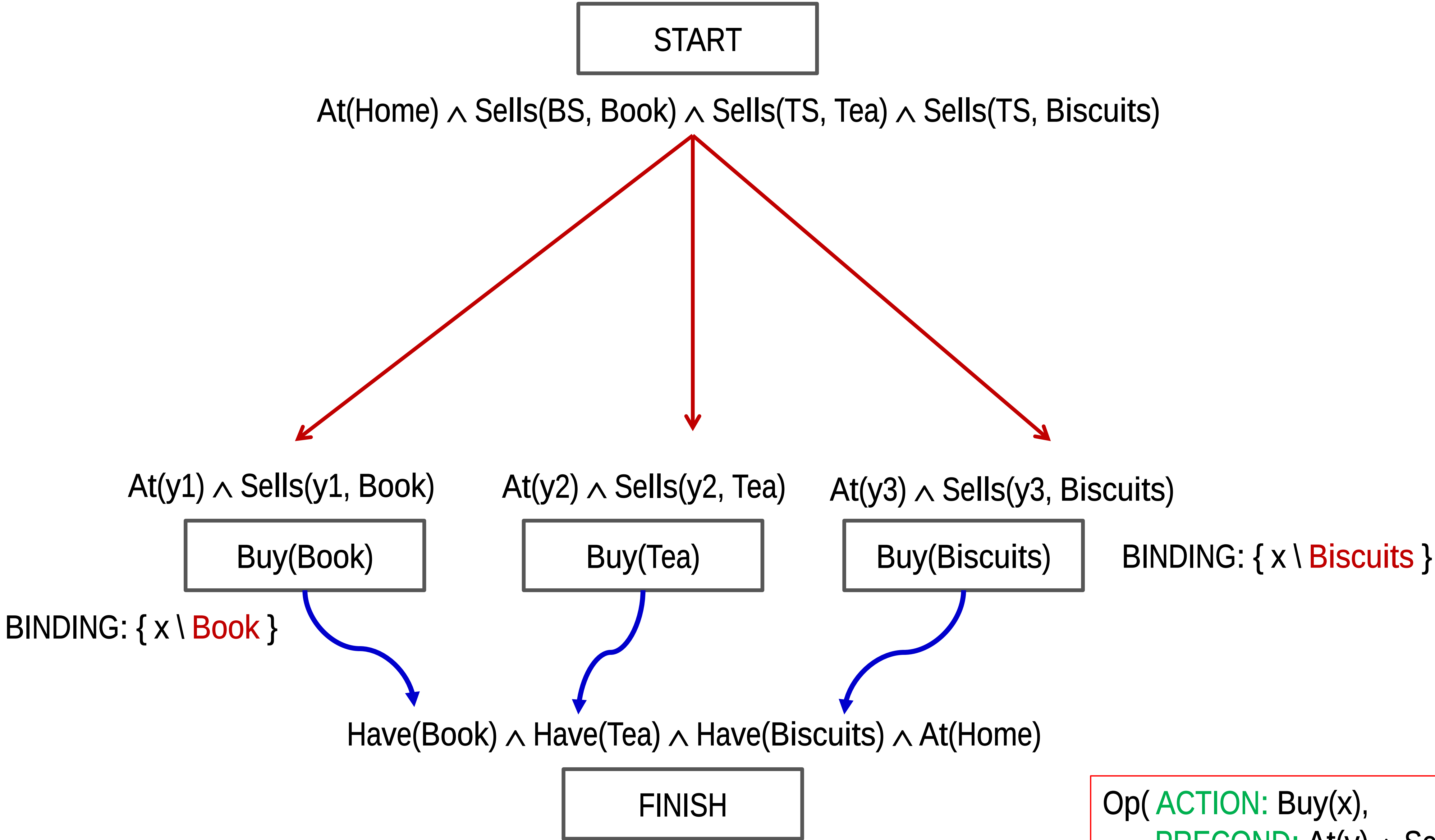
Buy(Book)

BINDING: { x \ Book }

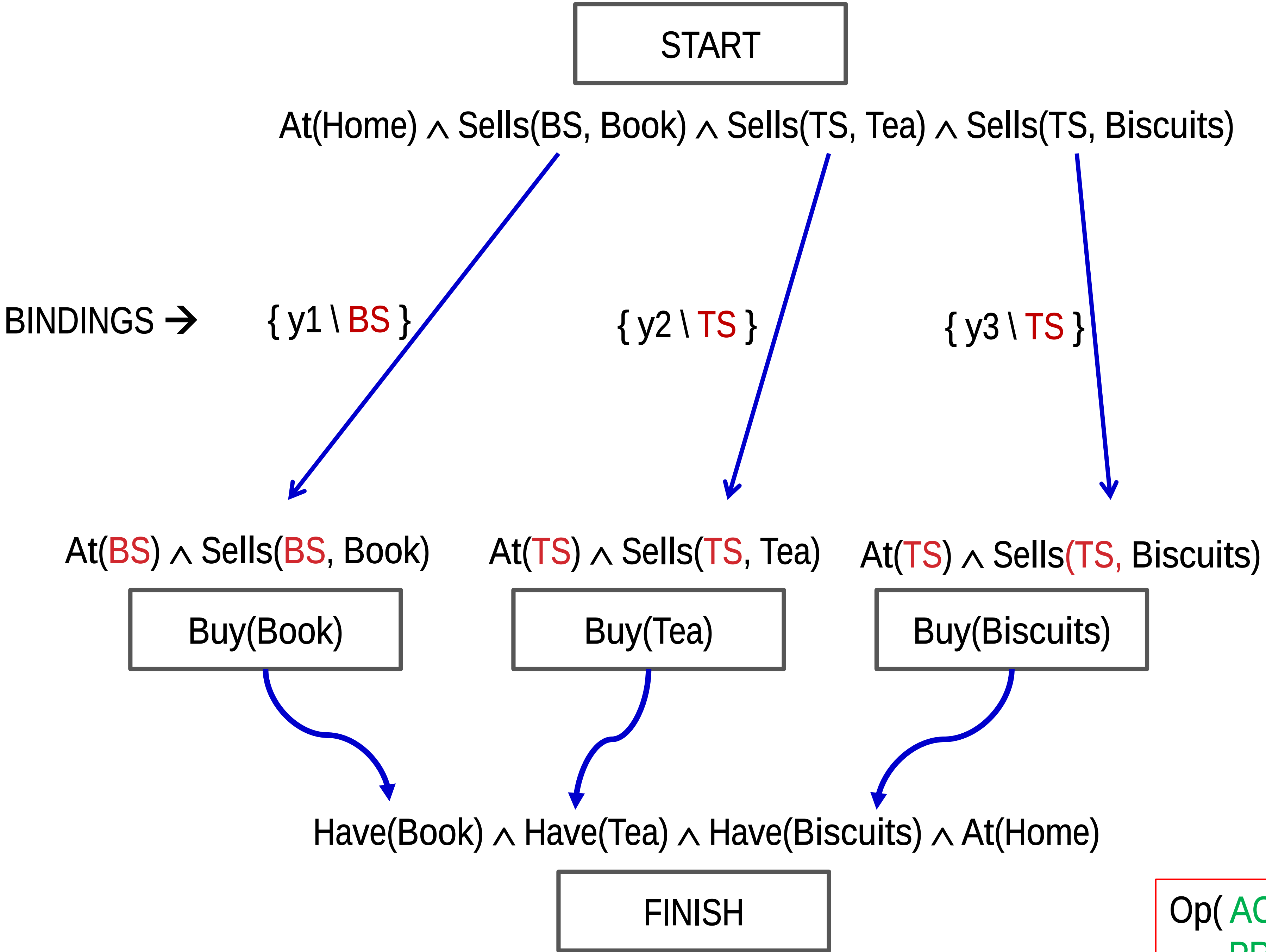
$Have(Book) \land Have(Tea) \land Have(Biscuits) \land At(Home)$

FINISH

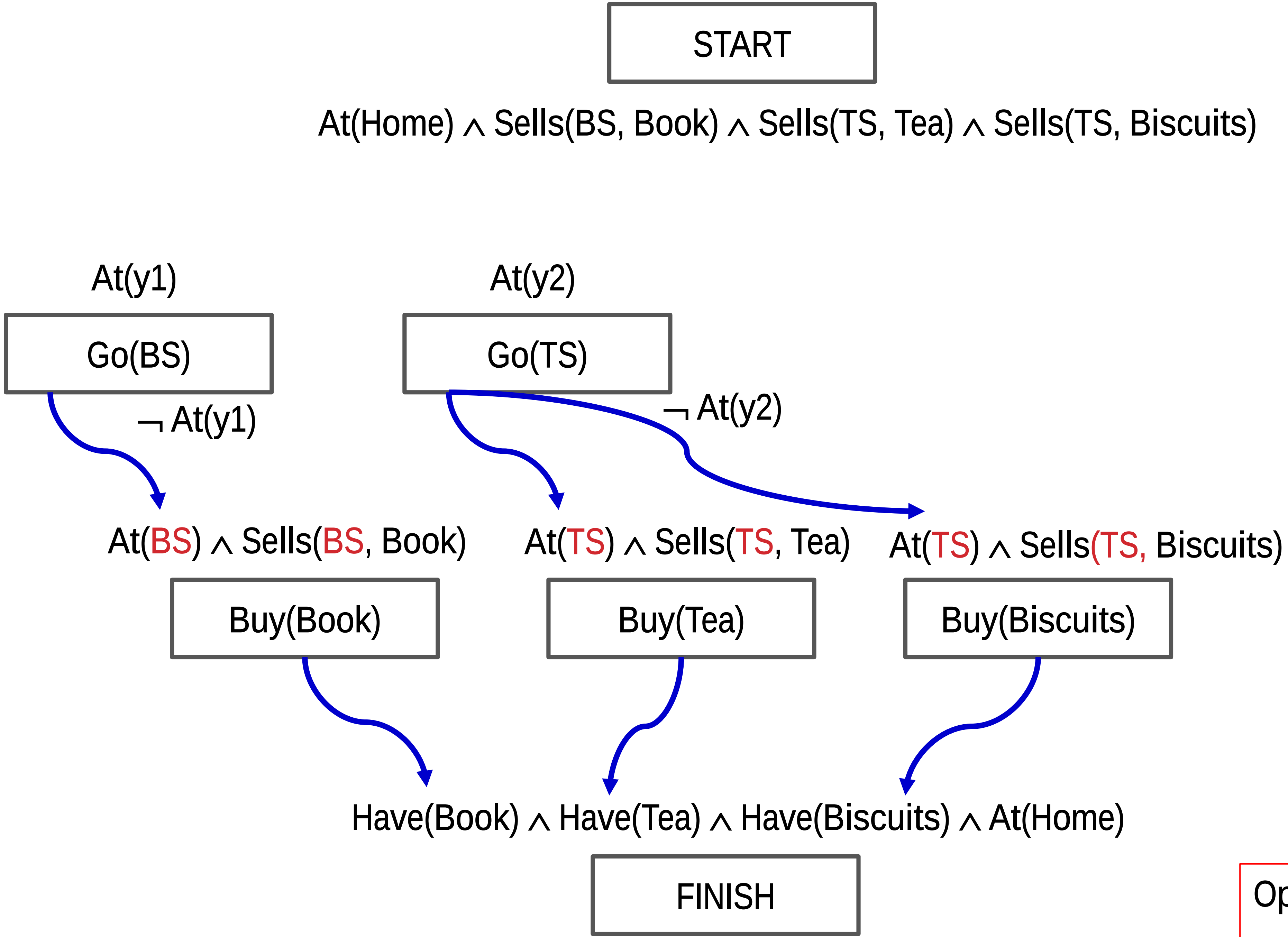
Op(ACTION: Buy(x),
PRECOND: $At(y) \land Sells(y, x)$,
EFFECT: Have(x))



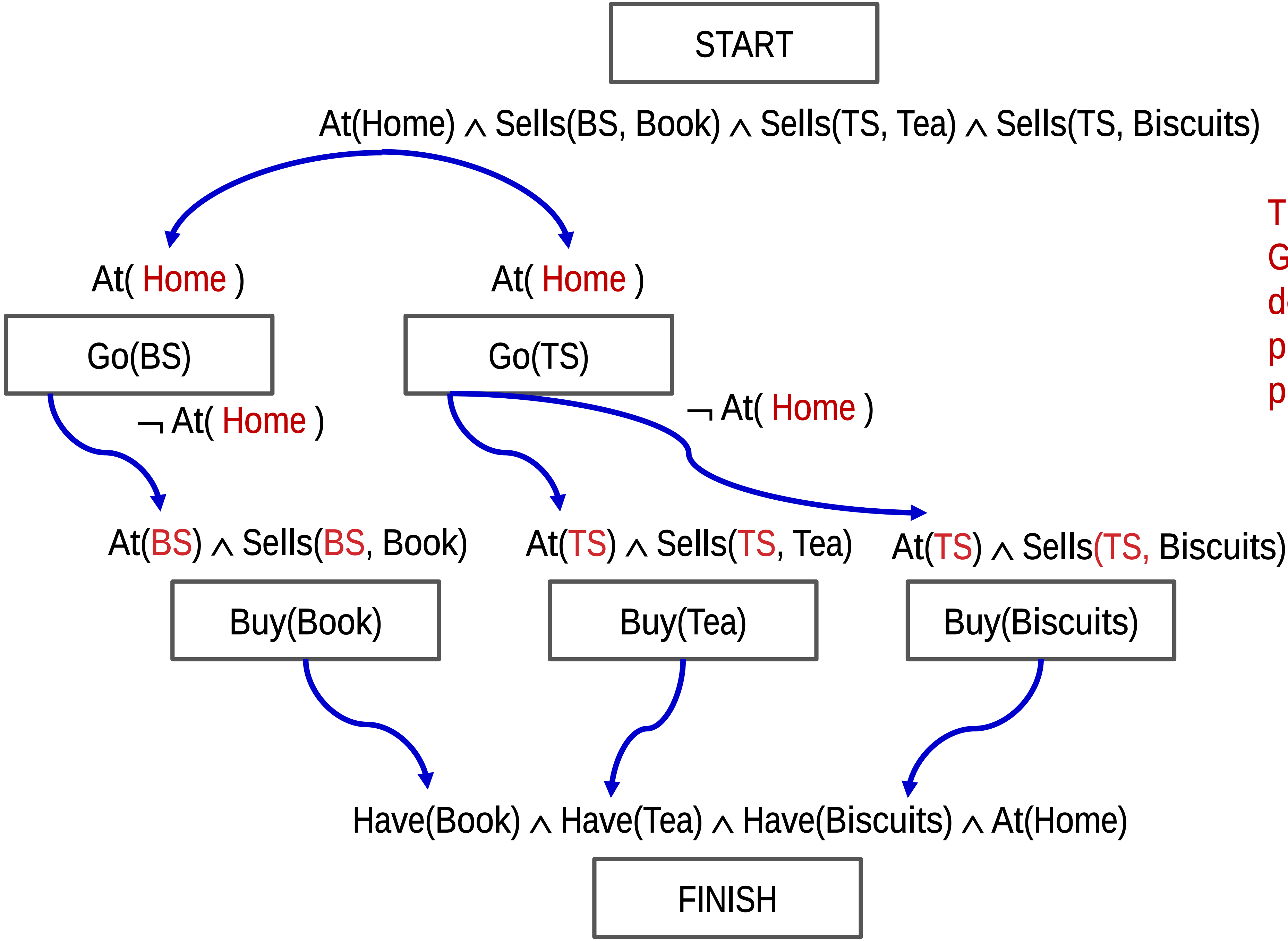
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))



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.

POP: Resolving Threats

Procedure Resolve-Threats(*plan*)

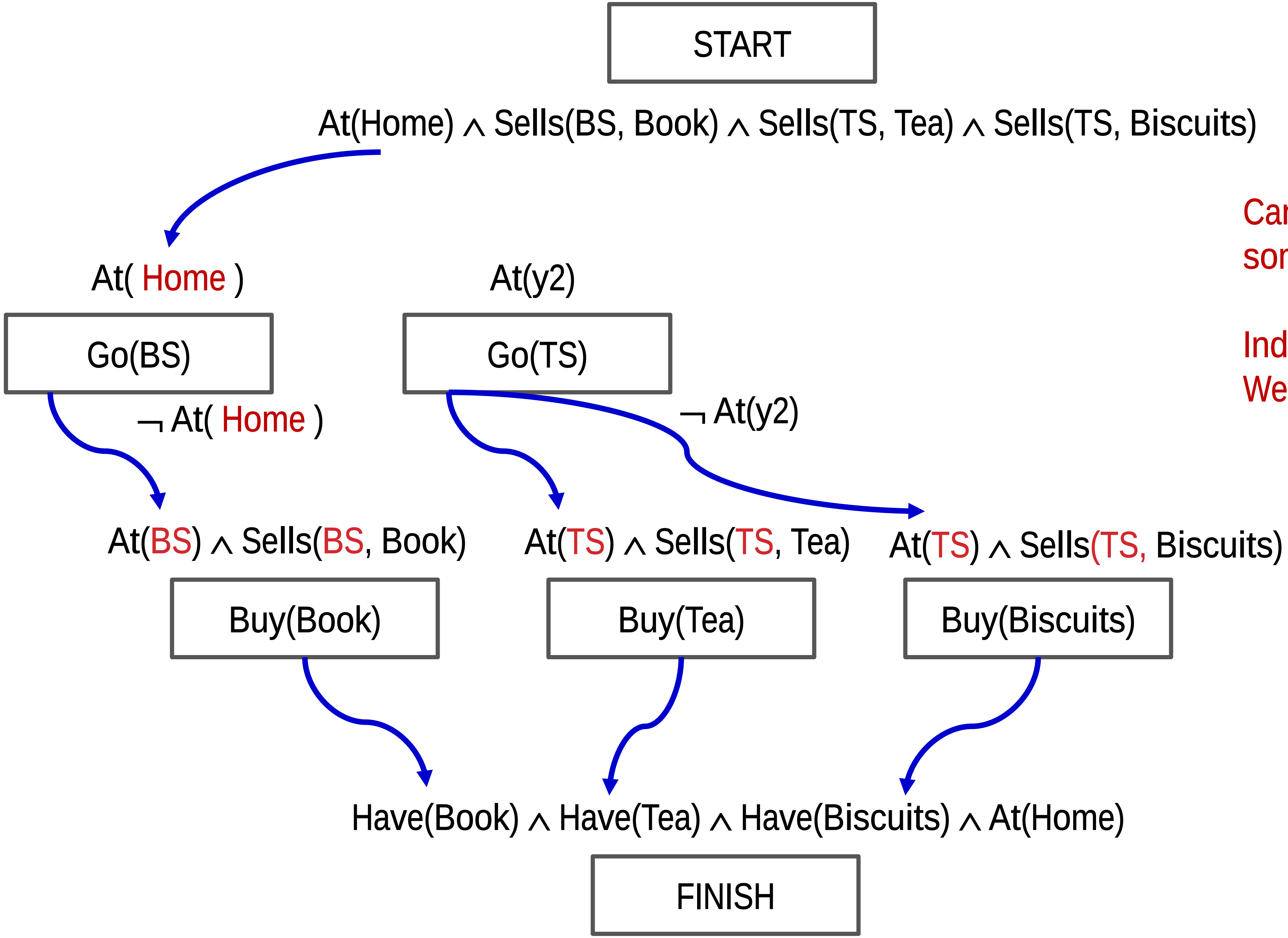
for each S' that threatens a link $S_i \rightarrow c: S_j$ in LINKS(*plan*) do

choose either

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

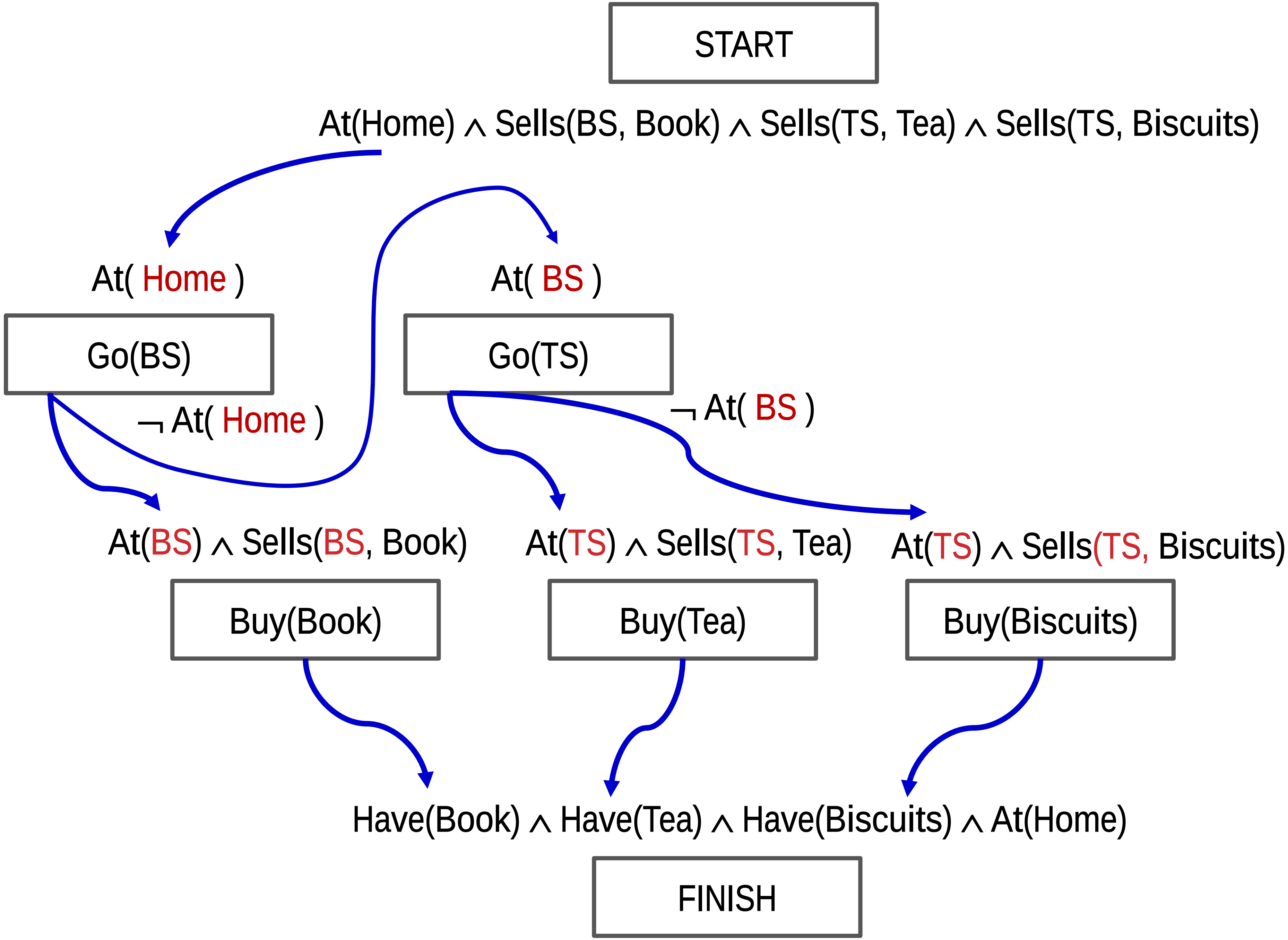
Demotion: Add $S_j \prec S''$ to ORDERINGS(*plan*)

if not Consistent(*plan*) then fail



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

POP: Resolving Threats

Procedure Resolve-Threats(*plan*)

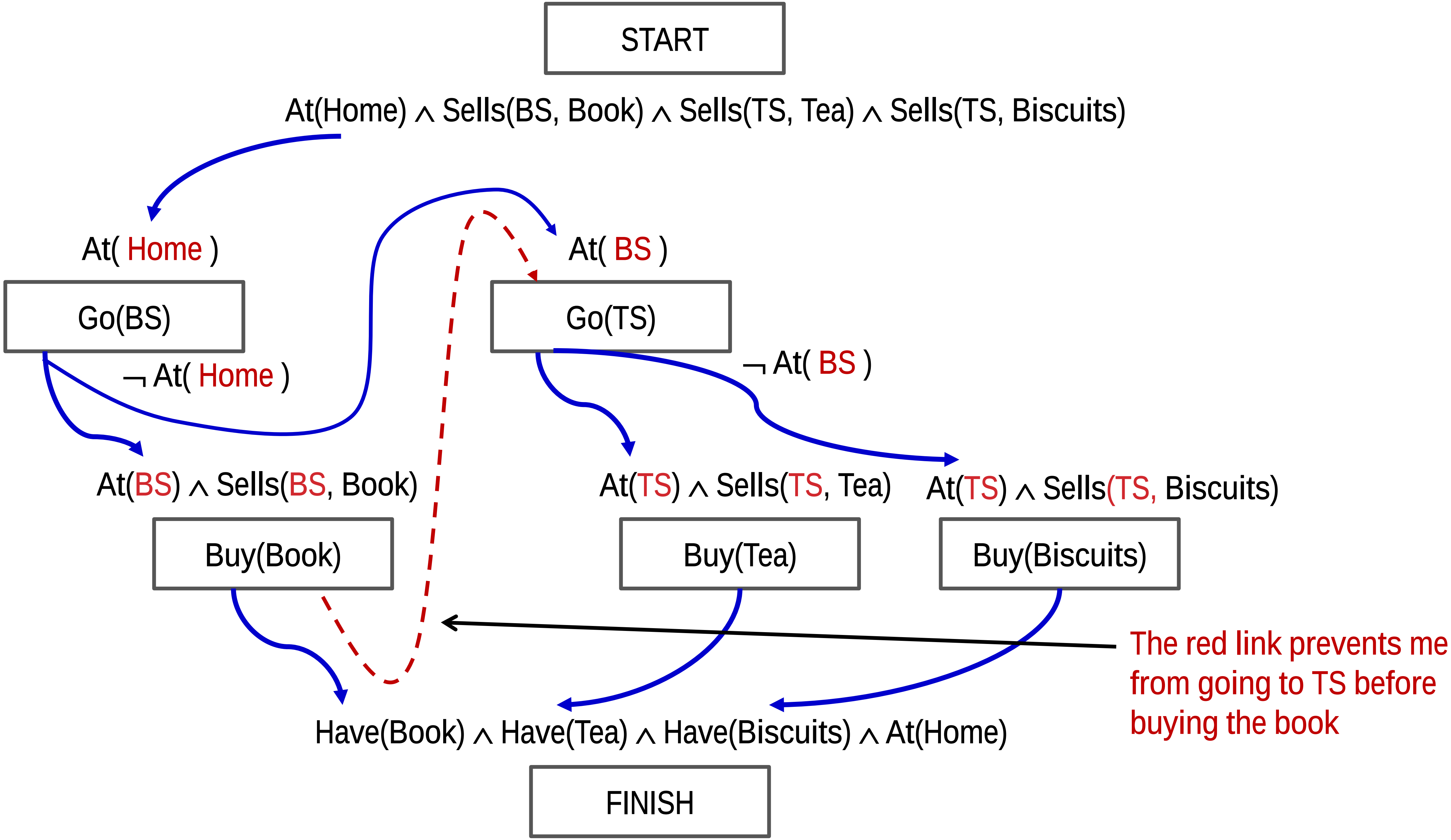
for each S' that threatens a link $S_i \rightarrow c: S_j$ in LINKS(*plan*) do

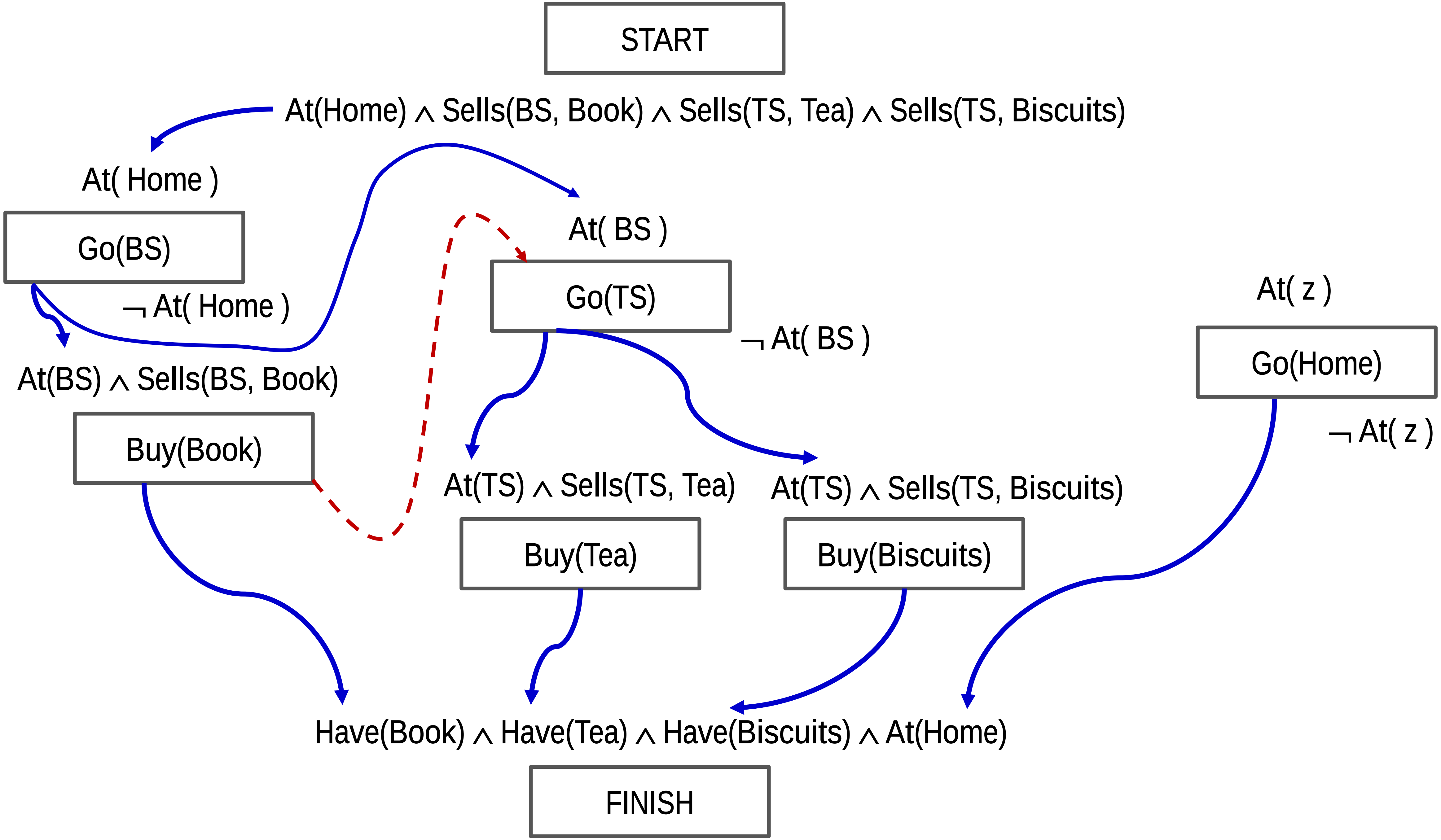
choose either

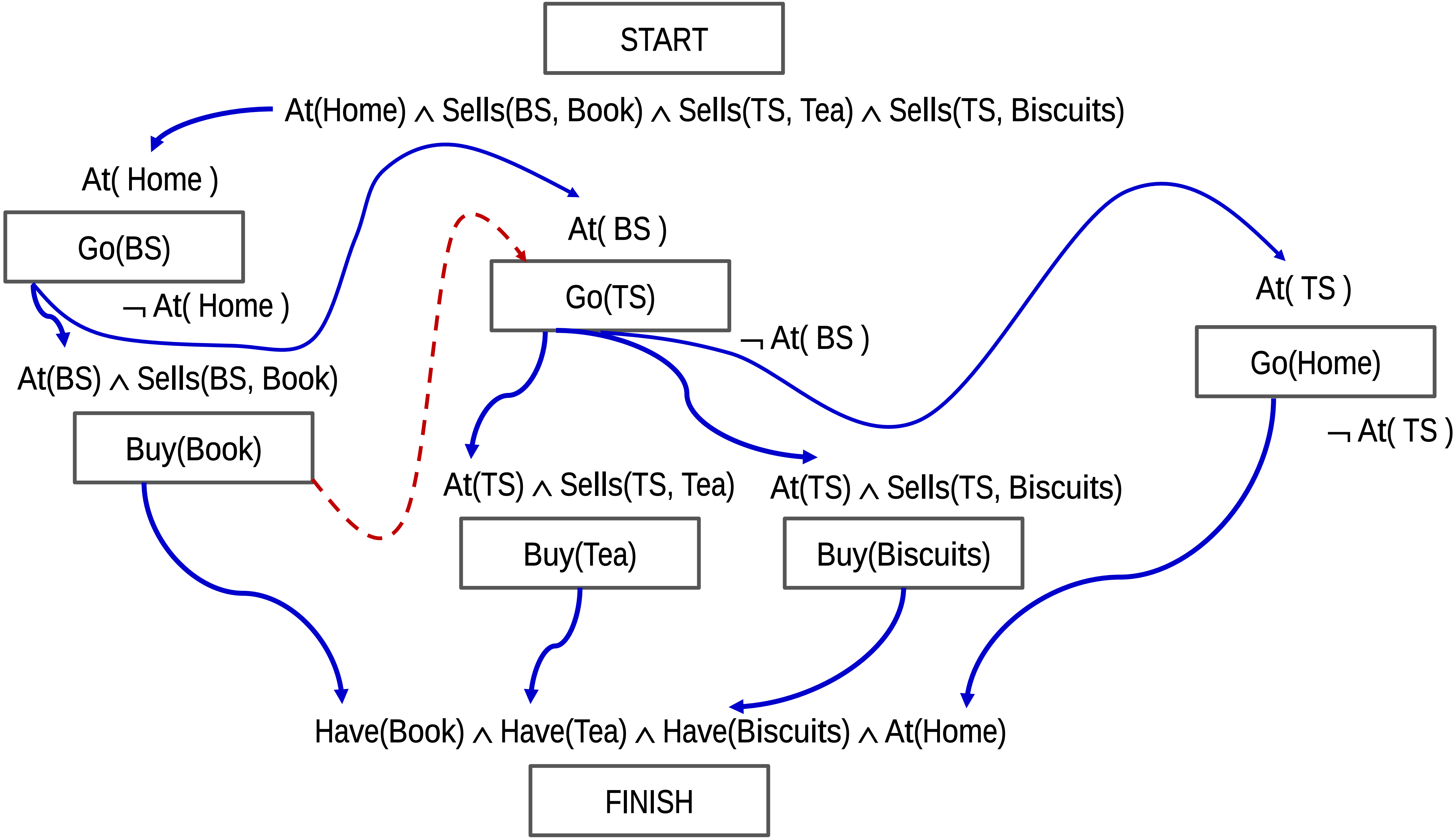
Promotion: Add $S'' \prec S_i$ to ORDERINGS(*plan*)

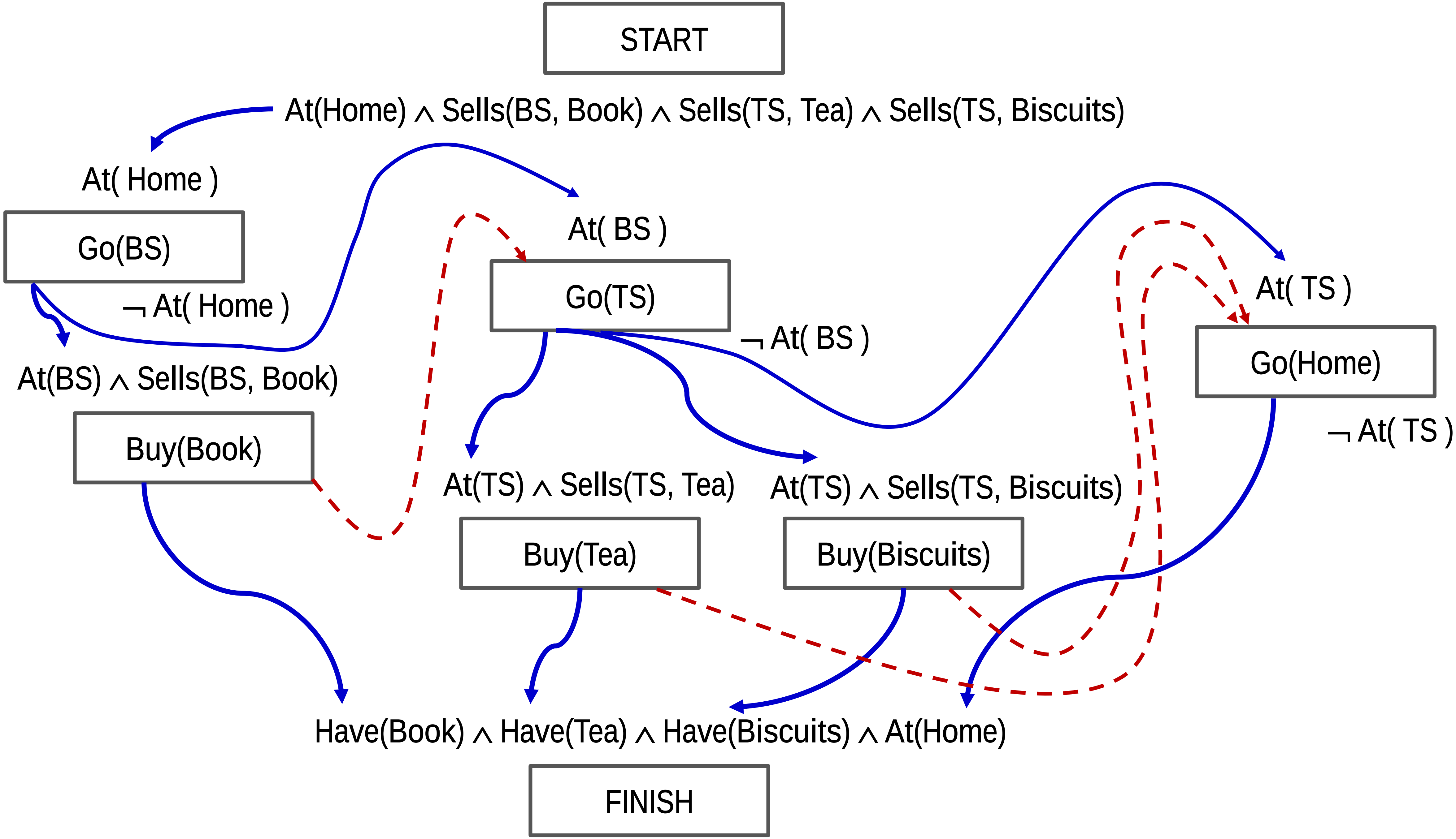
Demotion: Add $S_j \prec S''$ to ORDERINGS(*plan*)

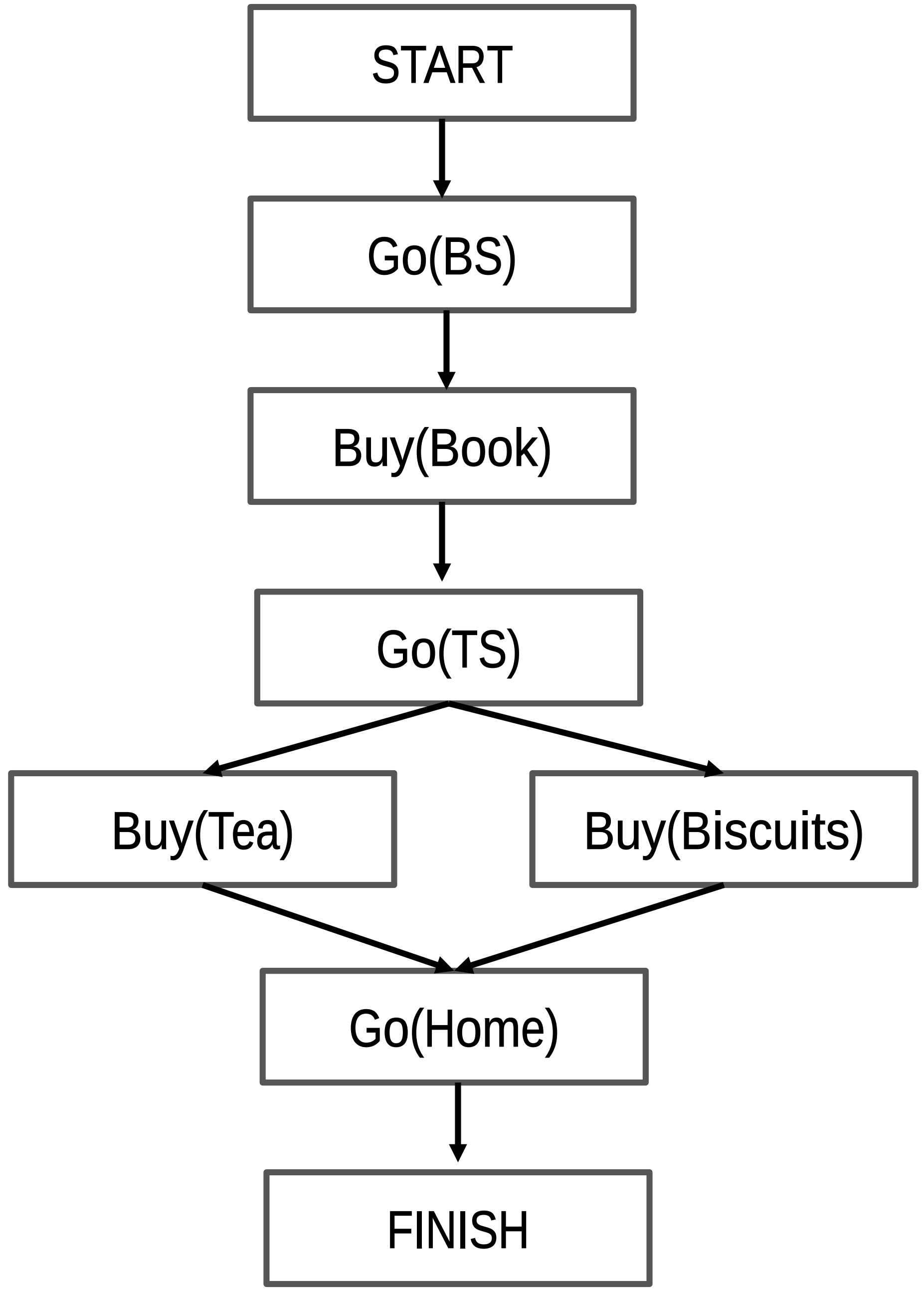
if not Consistent(*plan*) then fail


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

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 $S_i \rightarrow c: S_j$ 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 $S'' \prec S_i$ to ORDERINGS(*plan*)

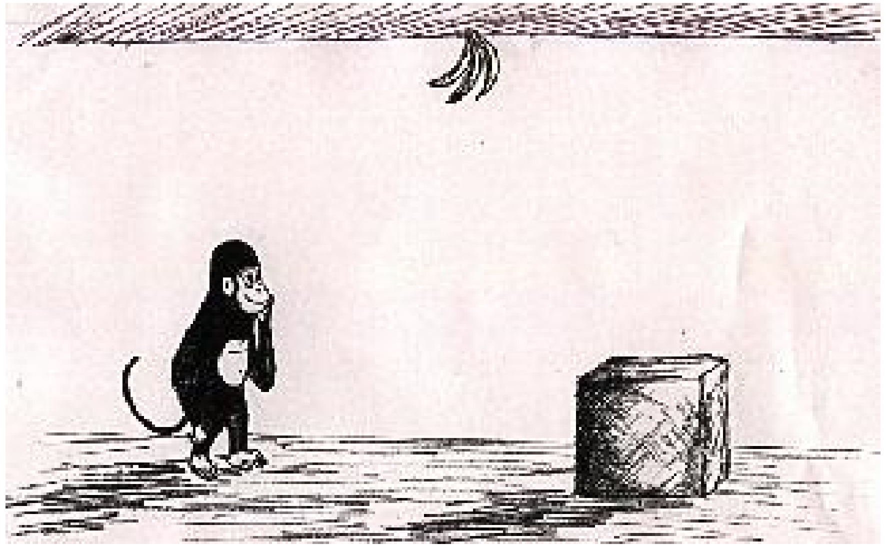
Demotion: Add $S_j \prec S''$ to ORDERINGS(*plan*)

if not Consistent(*plan*) then fail

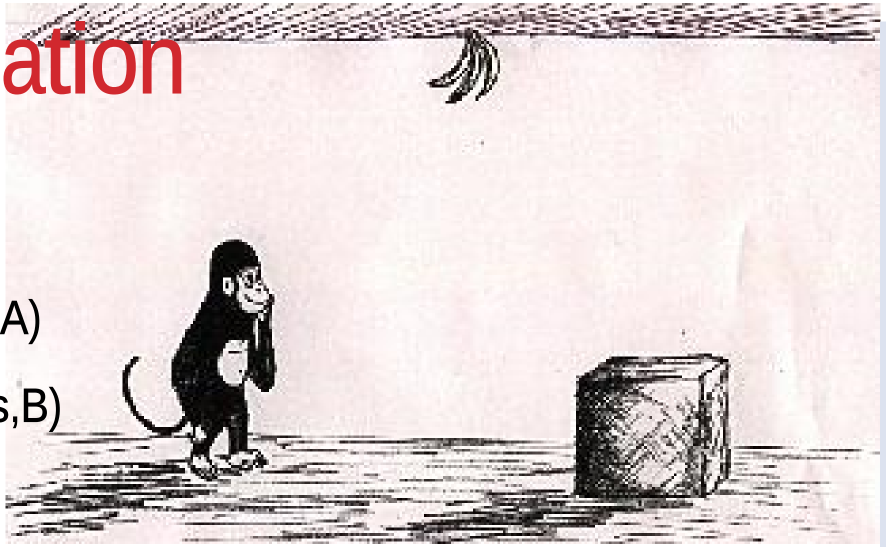
Monkey Bananas Problem

Assume that there is a monkey in a room with some bananas hanging out of reach from the ceiling, but a box is available that will enable the monkey to reach the bananas if he climbs on it.

- Initially, the monkey is at A, the bananas at B, and the box at C.
- The monkey and box have height LOW, but if the monkey climbs onto the box, he will have height HIGH, the same as the bananas.
- The actions available to the monkey include GO from one place to another, PUSH an object from one place to another, CLIMB onto an object, and GRASP an object. Grasping results in holding the object if the monkey and object are in the same place at the same height.
- The monkey wants to get the bananas.



Formulation



Initial State:

At(Monkey,A)

At(Bananas,B)

At(Box,C)

Height(Monkey,Low)

Height(Box,Low)

Height(Bananas,High)

Pushable(Box)

Climbable(Box)

Graspable(Bananas)

Goal State:

Have(Monkey, Bananas)

Operators:

Go(x,y)

Precond: At(Monkey,x) AND Height(Monkey,Low)

Effect: At(Monkey,y) AND NOT At(Monkey,x)

Push(b,x,y)

Precond: At(Monkey,x) AND Height(Monkey,Low) AND At(b,x) AND Pushable(b) AND Height(b,Low)

Effect: At(b,y) AND At(Monkey,y) AND NOT At(b,x) AND NOT At(Monkey,x)

ClimbUp(b)

Precond: At(Monkey,x) AND Height(Monkey,Low) AND At(b,x) AND Climbable(x) AND Height(b,Low)

Effect: On(Monkey,b) AND NOT Height(Monkey,Low) AND Height(Monkey,High)

Grasp(b)

Precond: At(Monkey,x) AND Height(Monkey,h) AND At(b,x) AND Graspable(b) AND Height(b,h)

Effect: Have(Monkey,b)

Door Locking System in a Car

I wish to determine whether I can possibly lock myself out of my car.

- Many predicates – whether I am inside / outside, whether the key is with me, inside, or outside the car, how the car can be locked
- Using the classical key or key fob
- Using the lock arming feature

Known Adversarial Planning

Environment is the planner

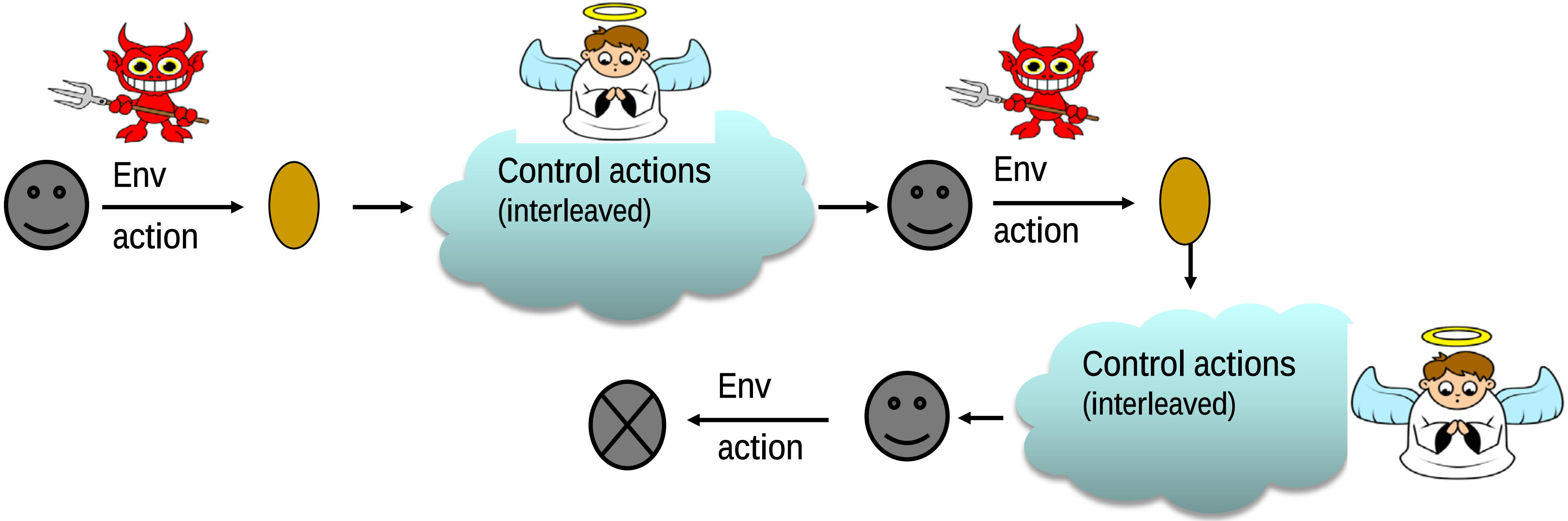
- It plans to drive the system to a bad state
- It has a set of actions to choose from
- It can choose to apply an action if its pre-condition is met

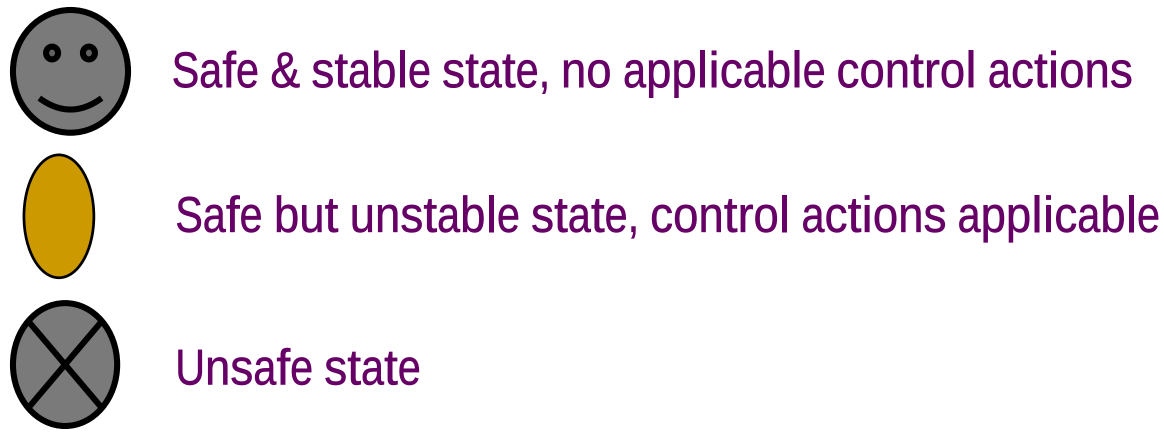
Controller is the adversary

- Distributed – comes from various sub-systems
- Predictable: Will apply whenever applicable
- Known a priori
- If multiple actions are applicable, then they may be applied in various sequences. The choice of the sequence is with the planner

Control actions have priority over environment actions. Environment gets a chance only when no applicable control actions remain.

Planning as Verification





Kamalesh Ghosh, Pallab Dasgupta and S. Ramesh, Automated Planning as an Early Verification Tool for Distributed Control, Journal of Automated Reasoning, 54 (1), 31-68, 2015.