Introduction to Planning

COURSE: CS40002

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Outline

- ❑ Planning versus Search
- ❑ Representation of planning problems
    - Situation calculus
    - STRIPS
    - ADL
- ❑ Planning Algorithms
    - Partial order planning
    - GraphPlan
    - SATPlan

The Planning Problem

Get tea, biscuits, and a book.

- **Given:**
  - **Initial state:** **The agent is at *home* without tea, biscuits, book**
  - **Goal state:** **The agent is at *home* with tea, biscuits, book**

The Planning Problem

- States can be represented by predicates such as At(x), Have(y), Sells(x, y)

- Actions:
  - Go(y) : Agent goes to y
    – causes At(y) to be true
  - Buy(z): Agent buys z
    – causes Have(z) to be true
  - Steal(z): Agent steals z

Planning as Search

- Actions are given as logical descriptions of preconditions and effects.
    - This enables the planner to make direct connections between states and actions.
- The planner is free to add actions to the plan wherever they are required, rather than in an incremental way starting from the initial state.
- Most parts of the world are independent of most other parts – hence divide & conquer works well.

Situation Calculus

- **Initial state:**

$$At(Home, s0) \land \neg Have(Tea, s0) \land \neg Have(Biscuits, s0) \land \neg Have(Book, s0)$$

- **Goal state:**

$$\exists s\ At(Home, s) \land Have(Tea, s) \land Have(Biscuits, s) \land Have(Book, s)$$

Situation Calculus

Operators:

$$\forall\, a,s\ Have(Tea, Result(a,s)) \Leftrightarrow$$
$$[(a = Buy(Tea) \wedge At(Tea\text{-}shop,s))$$
$$\vee\ (Have(Tea, s) \wedge a \neq Drop(Tea))]$$

Result(a,s) names the situation resulting from executing the action a in the situation s

Practical Planners

- To make planning practical we need to:

  - Restrict the language with which we define problems. With a restrictive language, there are fewer possible solutions to search through

  - Use a special-purpose algorithm called a *planner* rather than a general purpose theorem prover.

STRIPS

- STanford Research Institute Problem Solver

- Many planners today use specification languages that are variants of the one used in STRIPS.

Representing states

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

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

Representing goals

- Goals are also described by conjunctions of literals

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

- Goals can also contain variables

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

  - 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

- 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)

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: { })

Action Description Language (ADL)

STRIPS	ADL
Only +ve literals in states *Fat ∧ Slow*	Both +ve and –ve literals in states *¬Thin ∧ ¬Fast*
Closed World: Unmentioned literals are false	Open World: Unmentioned literals are unknown
Effect P∧¬Q means add P, delete Q	Effect P∧¬Q means add P, ¬Q and delete Q, ¬P

Action Description Language (ADL)

STRIPS	ADL
Only ground literals in goals *Fat* $\wedge$ *Slow*	**Quantified variables in goals** $\exists x$ At(Tea,x) $\wedge$ At(Coffee,x)
Goals are conjunctions	**Goals allow conjunctions and disjunctions**

Partial Order Planning

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))

Partial Order Planning

Actions:

Op(ACTION: Go(there),

PRECOND: At(here),

EFFECT: At(there) $\wedge$ $\neg$At(here))

Op(ACTION: Buy(x),

PRECOND: At(store) $\wedge$ Sells(store, x),

EFFECT: Have(x))

POP Example

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))

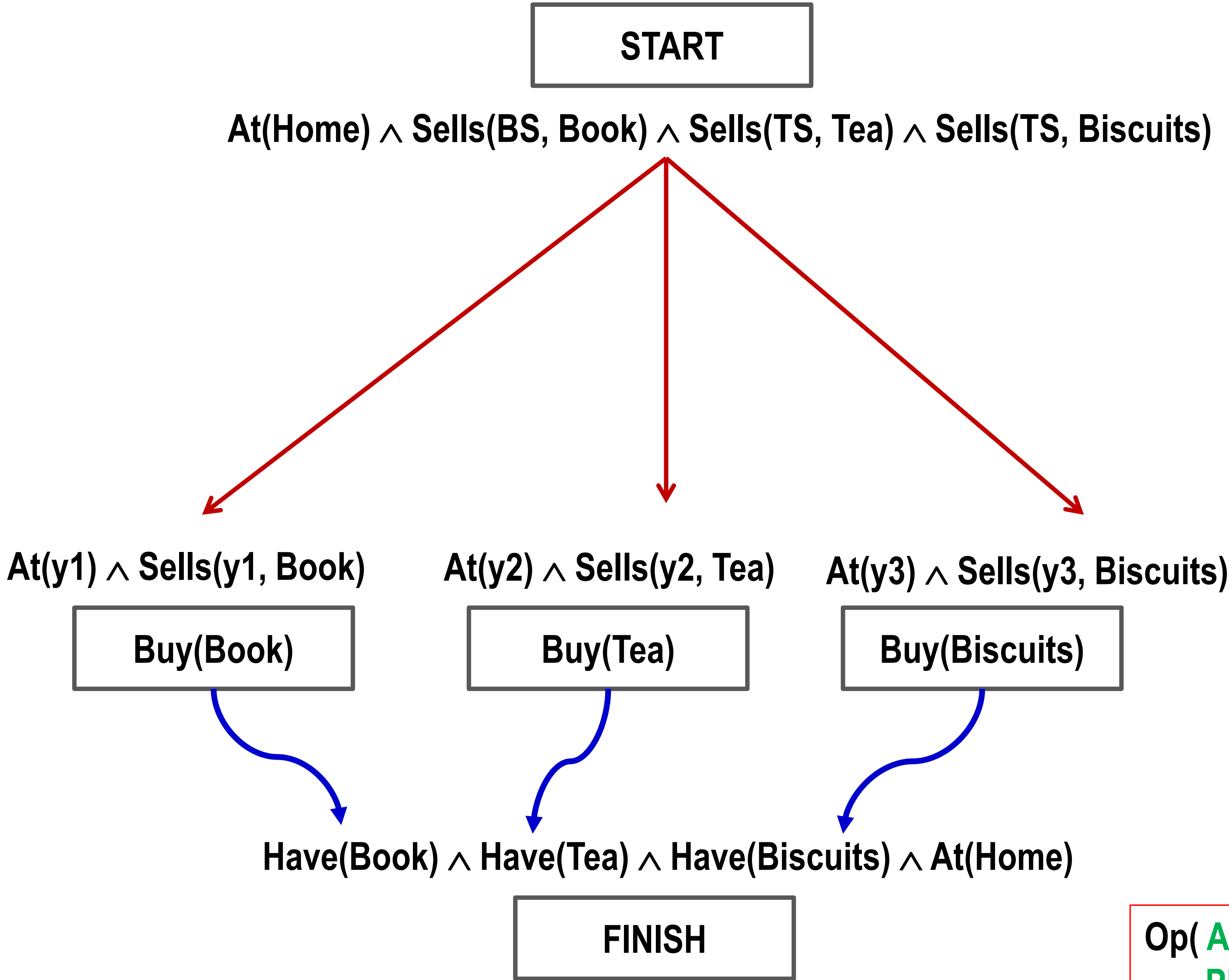
Op(ACTION: Buy(x),
  PRECOND: At(y) $\wedge$ Sells(y, x),
  EFFECT: Have(x))

START

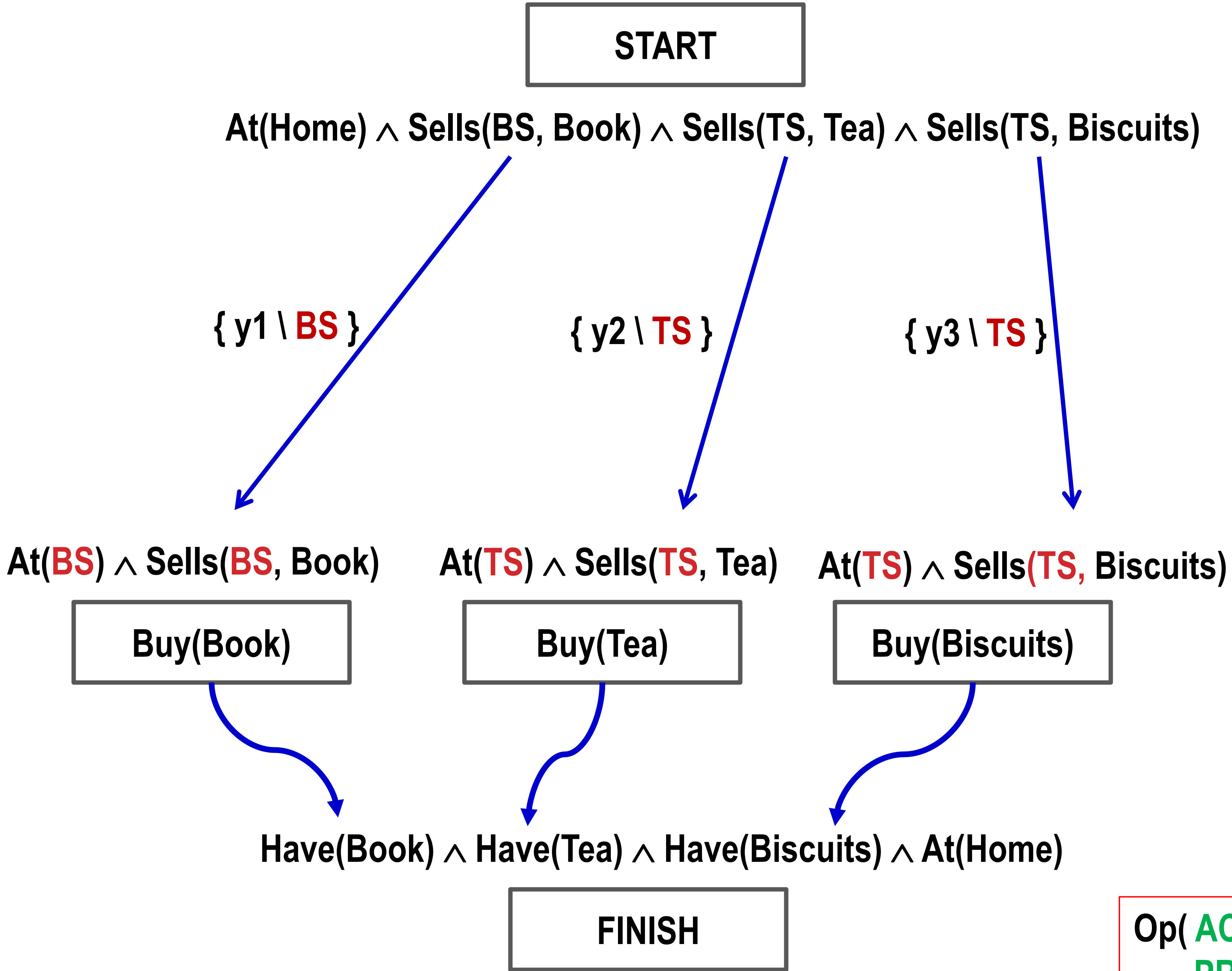
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)

Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)

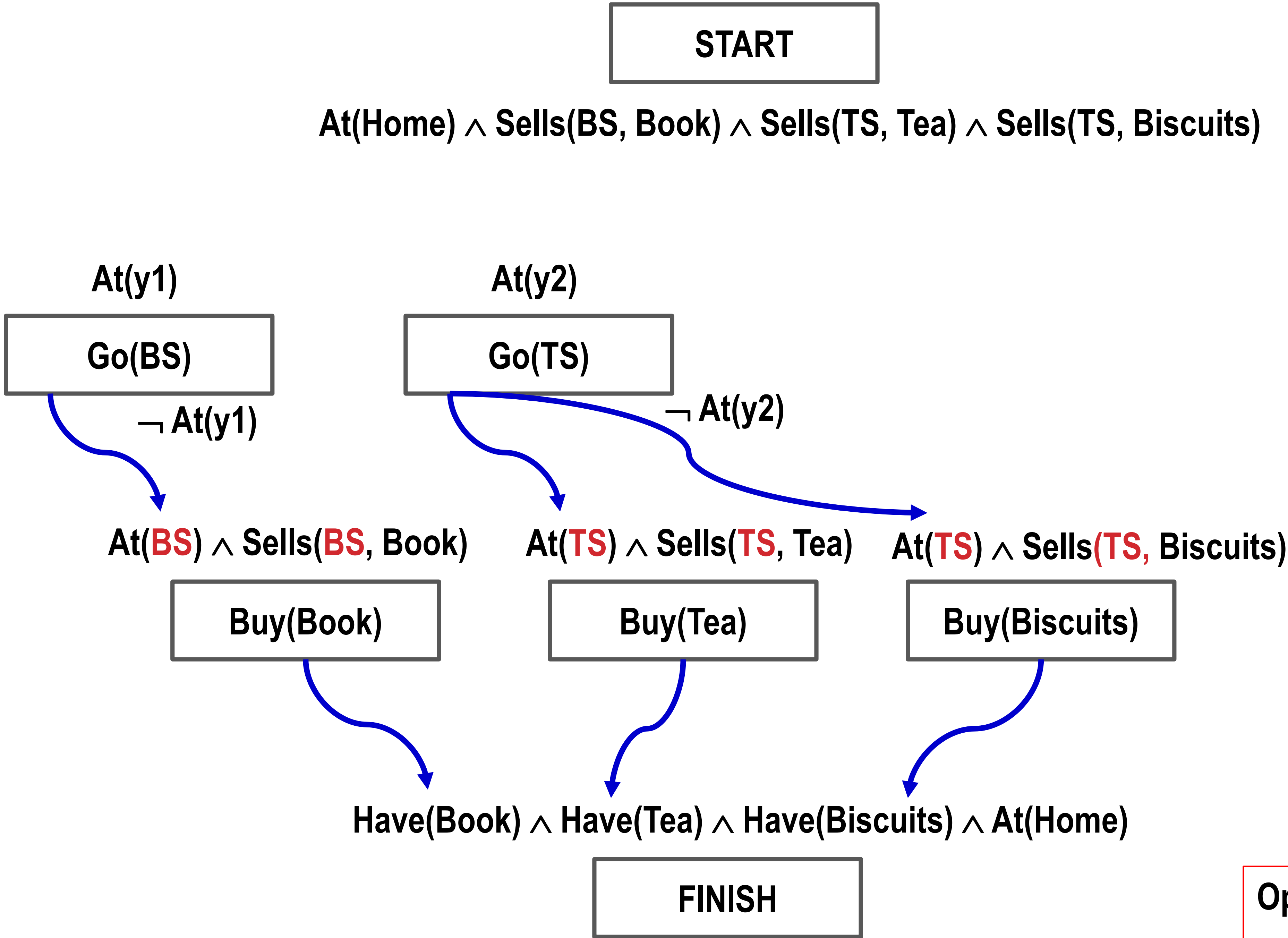
FINISH



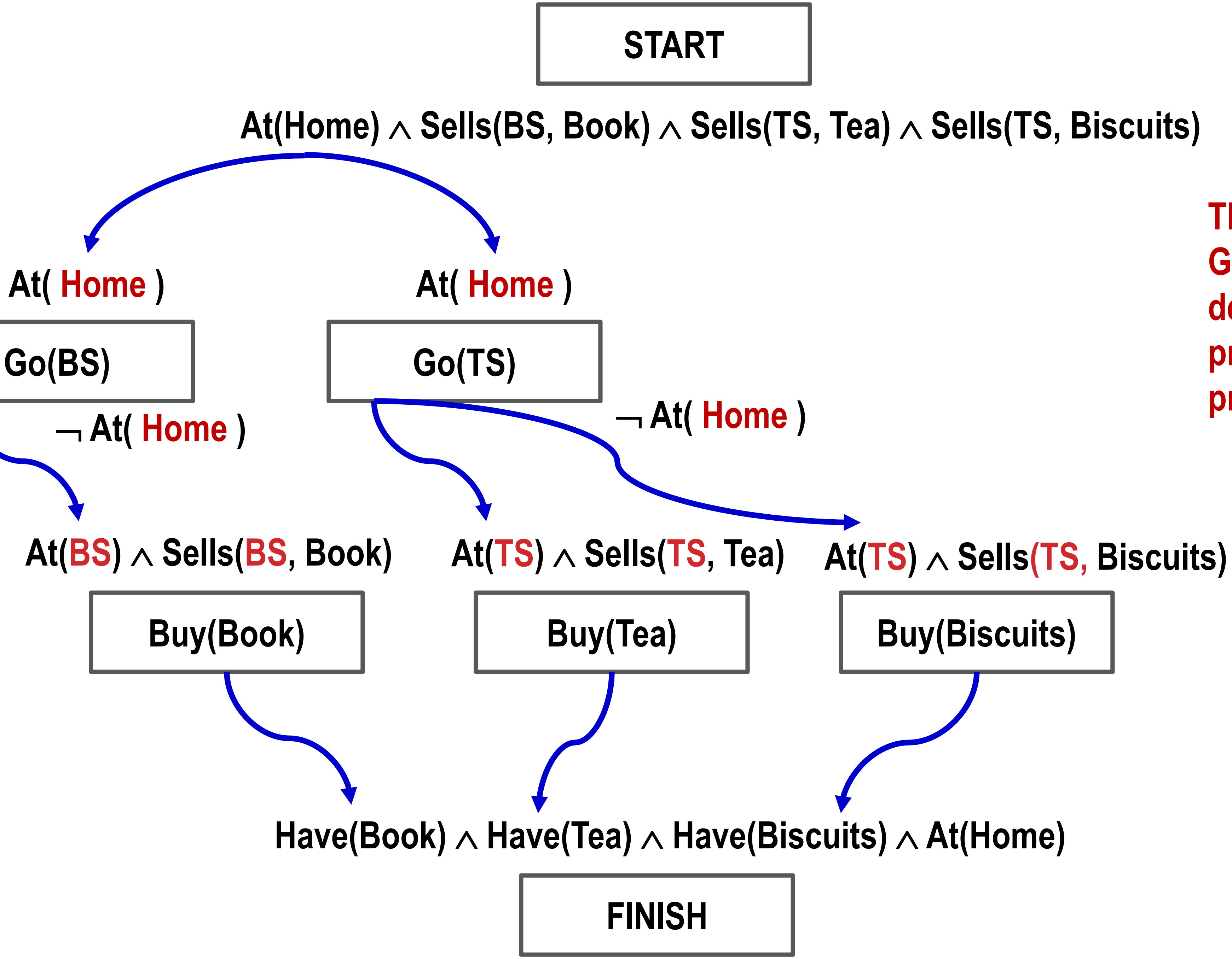
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))



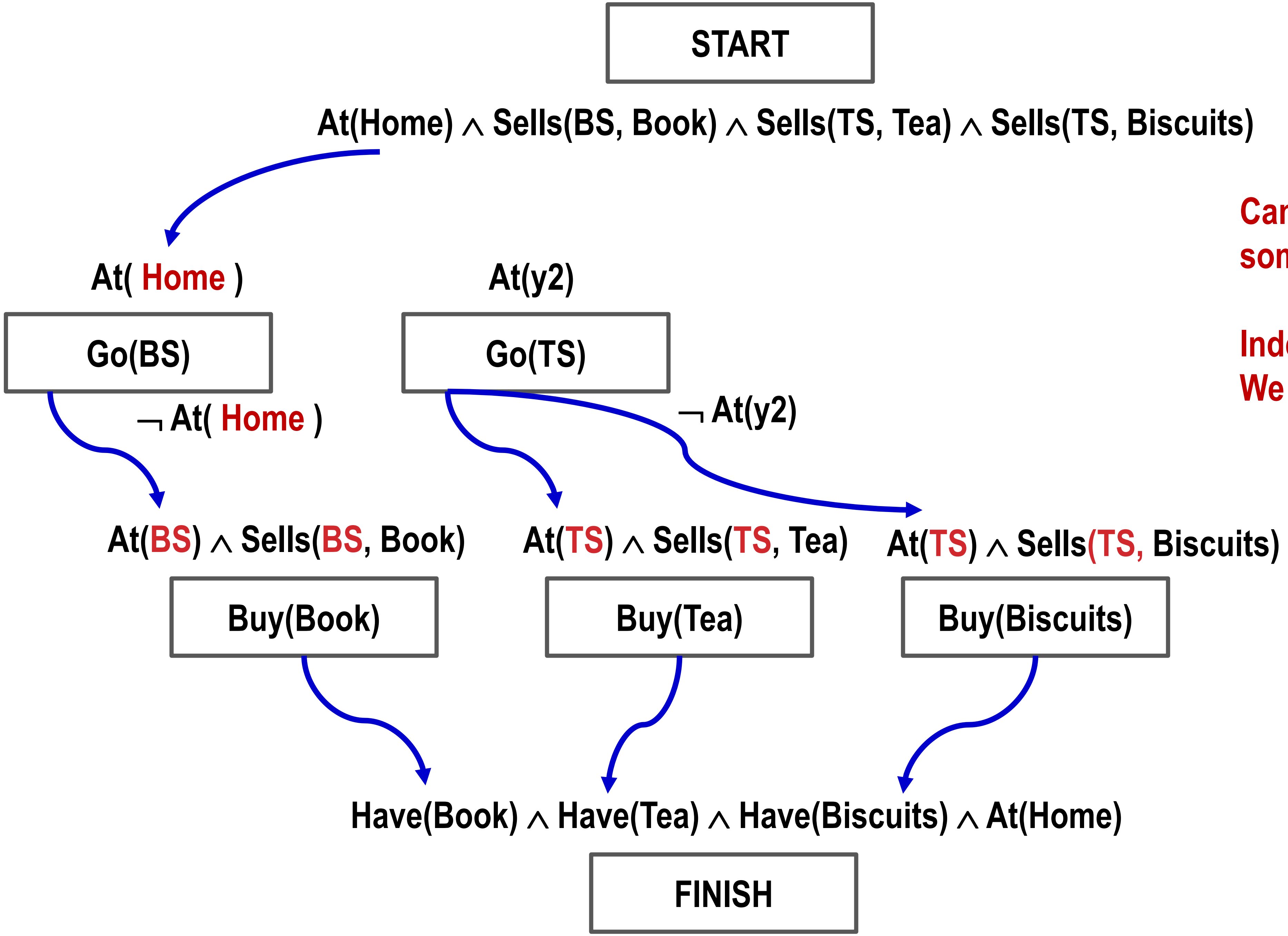
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))



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

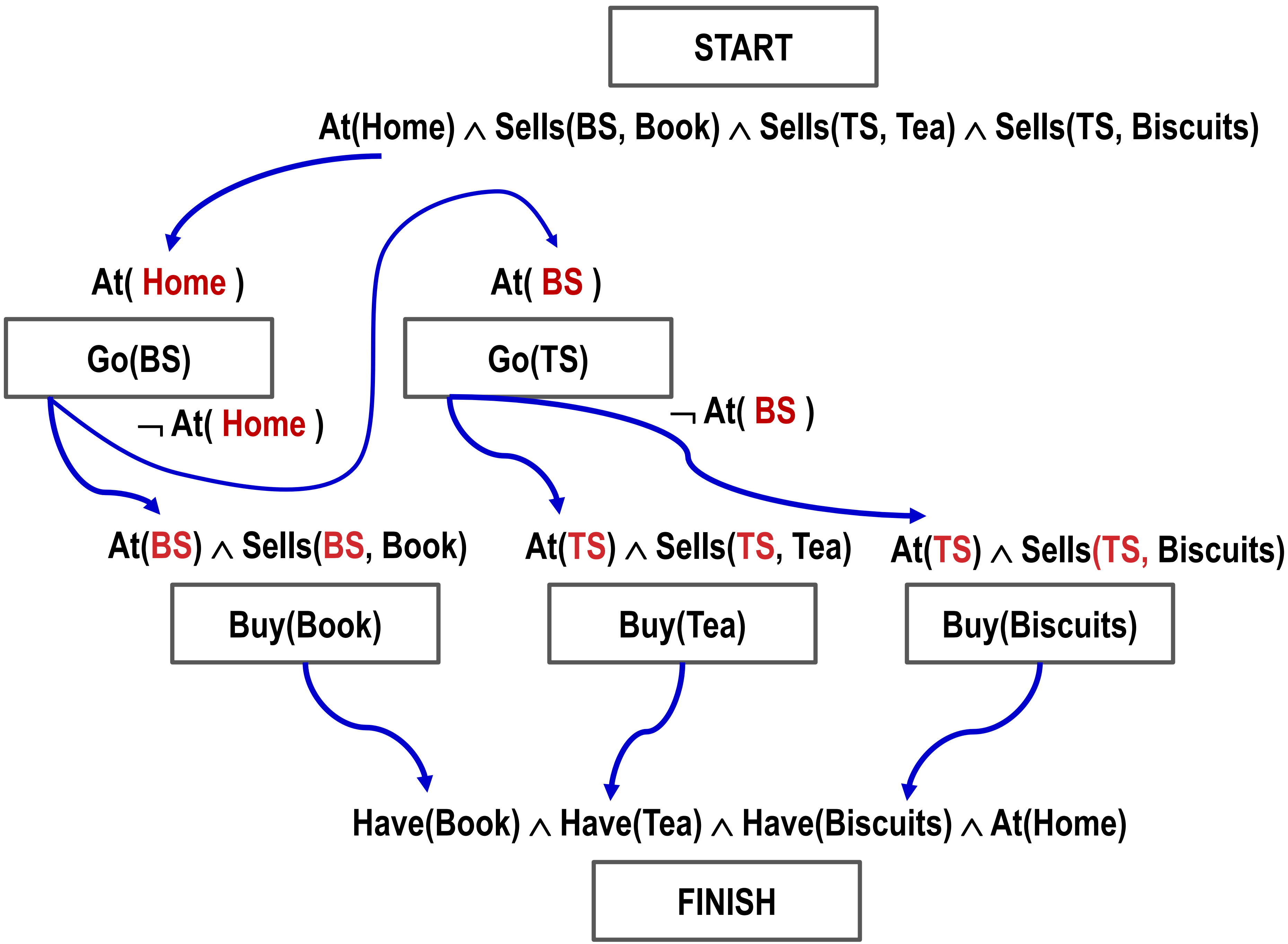


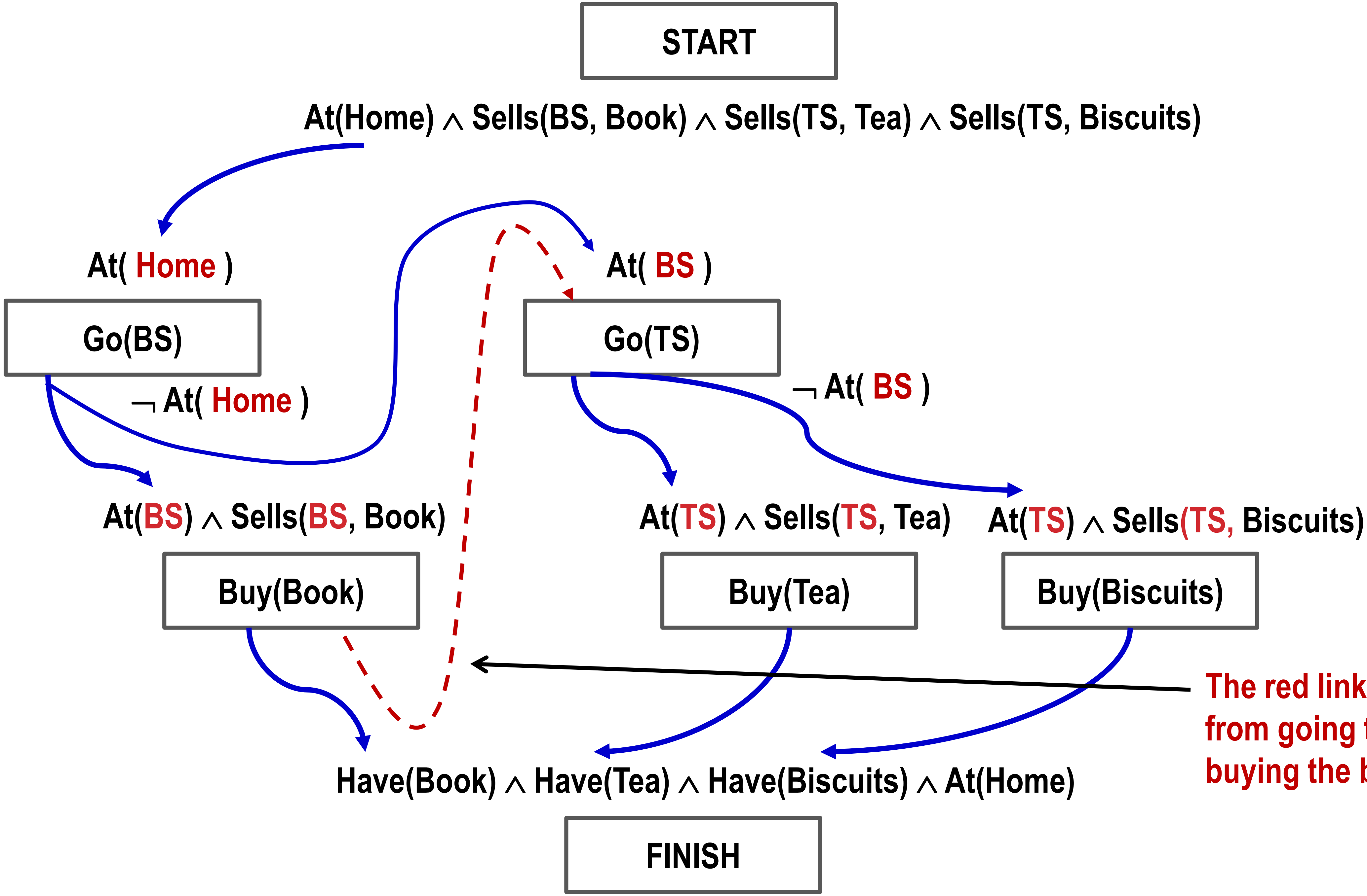
The problem here is that Go(BS) and Go(TS) destroy each other's precondition. Neither can precede the other.



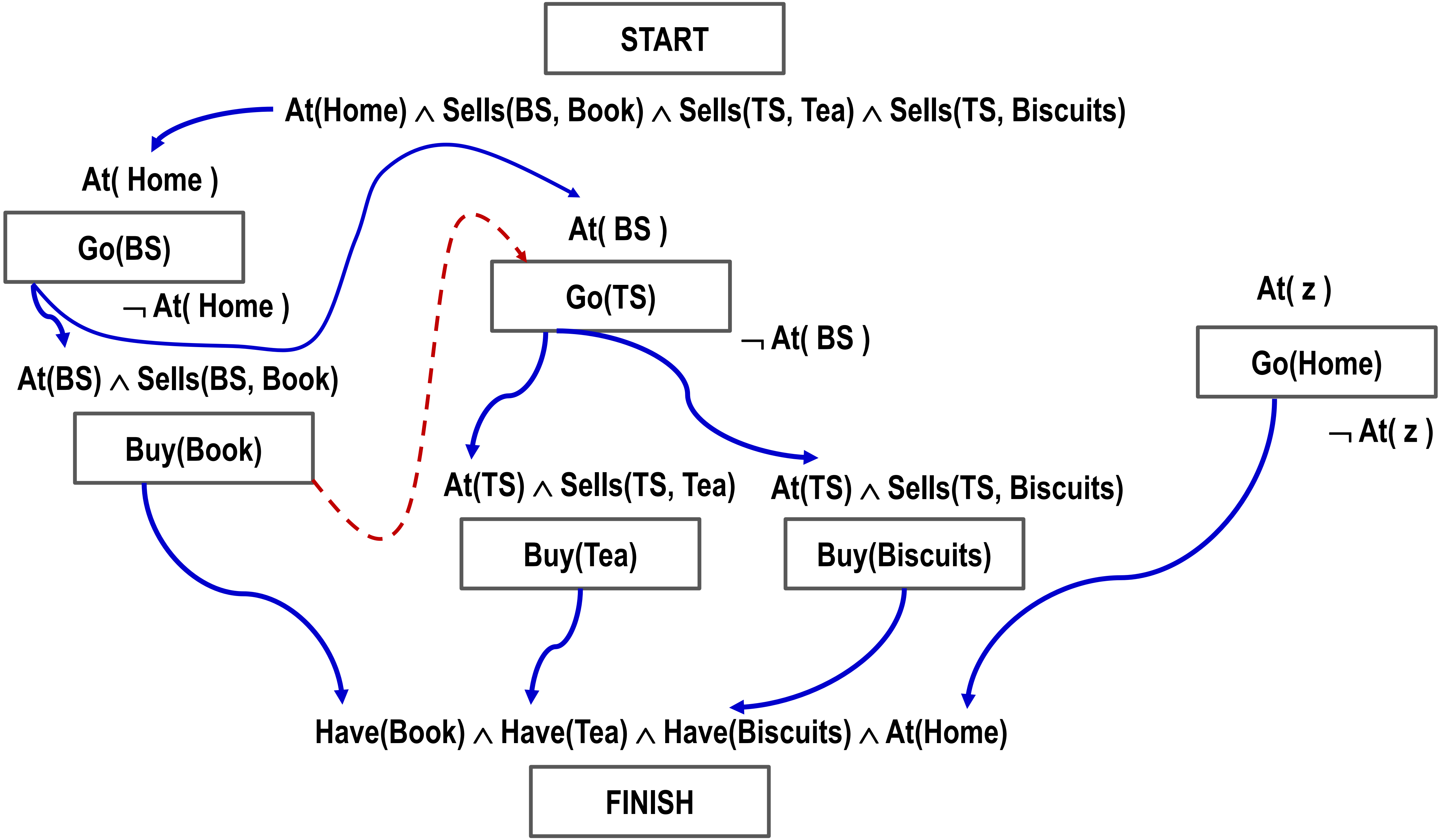
Can y2 be instantiated with something else?

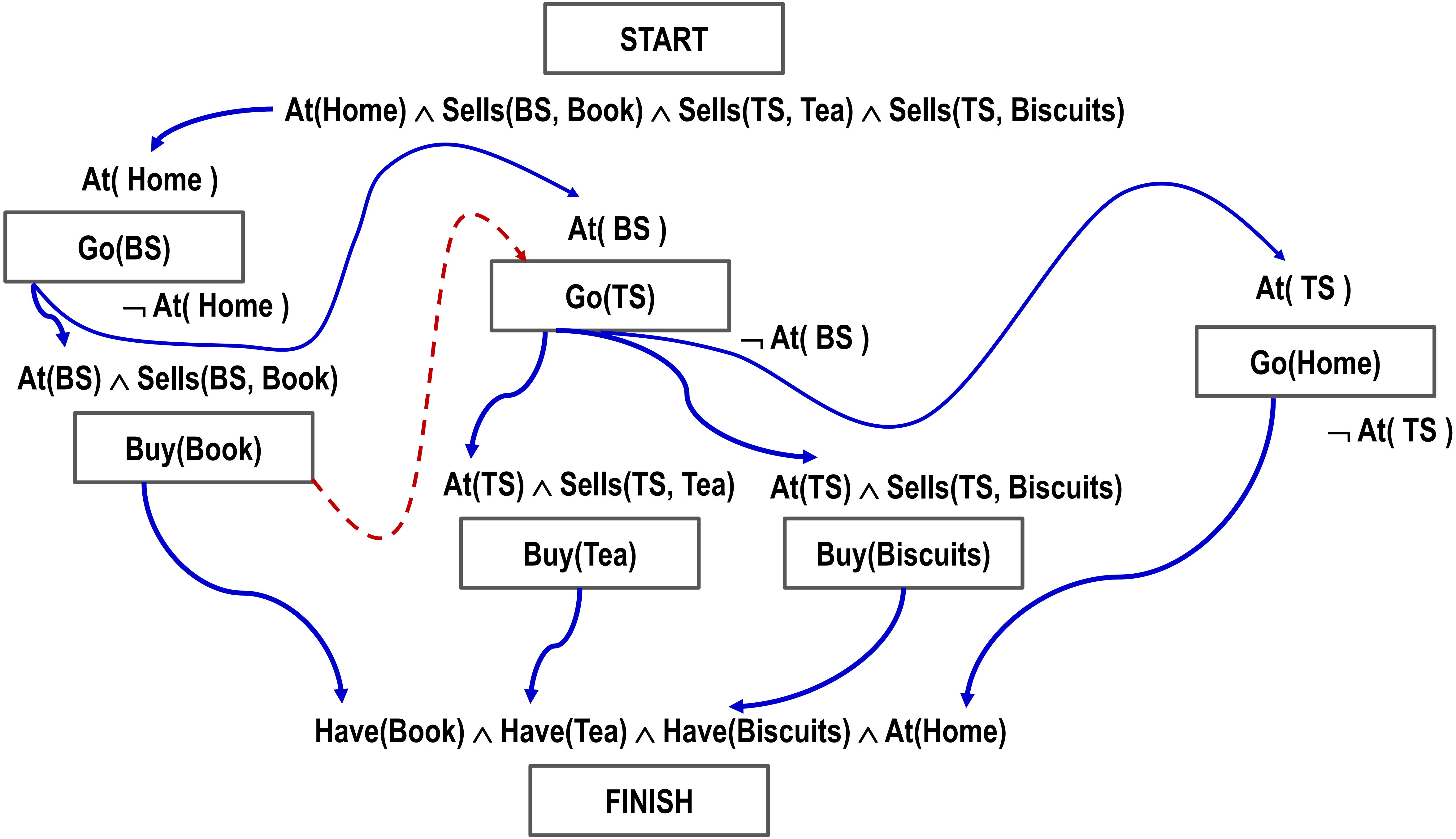
Indeed !!
We can try BS for example.

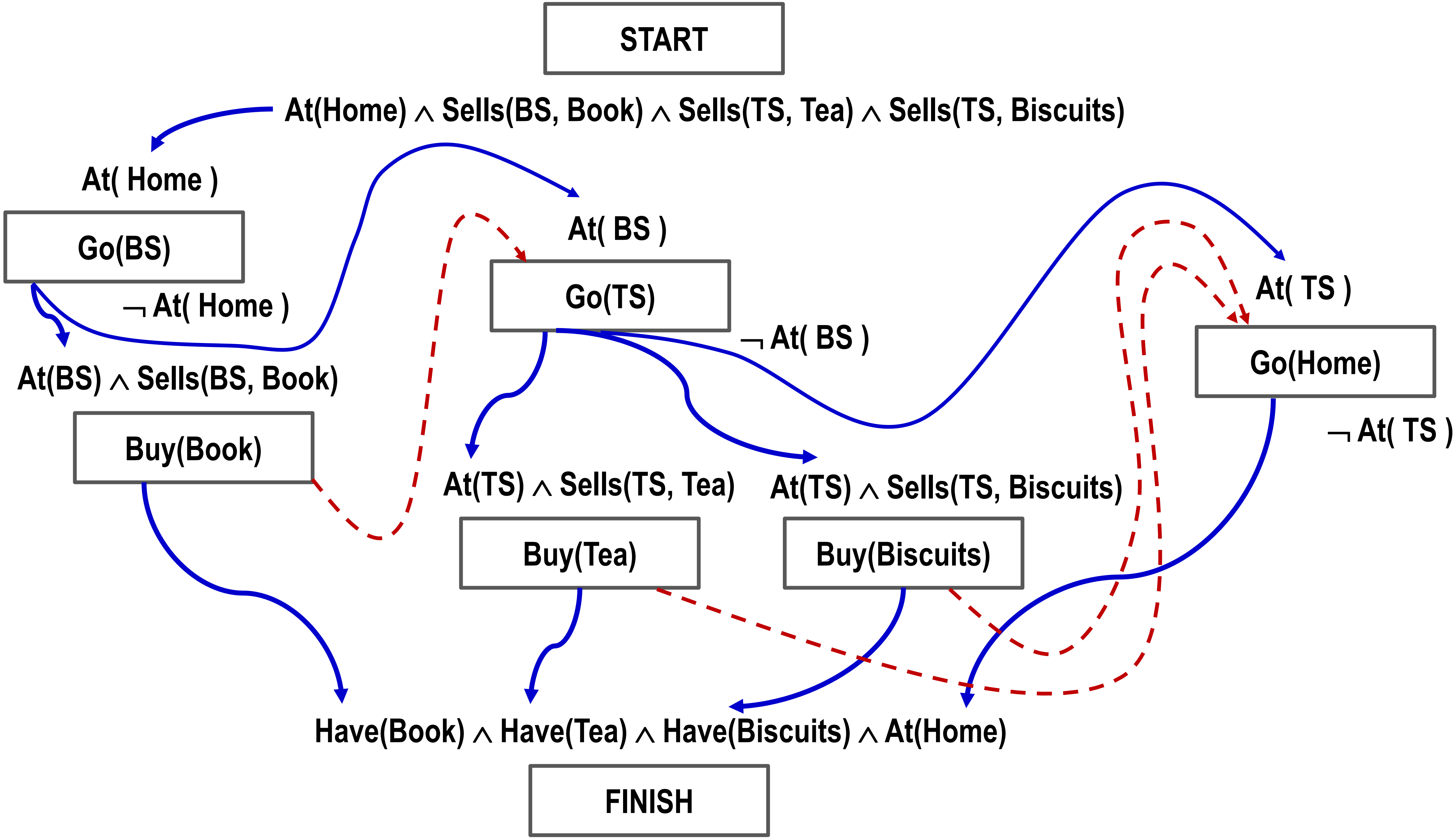

START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At(Home)
At(BS)
Go(BS)
Go(TS)
¬ At(Home)
¬ At(BS)
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH

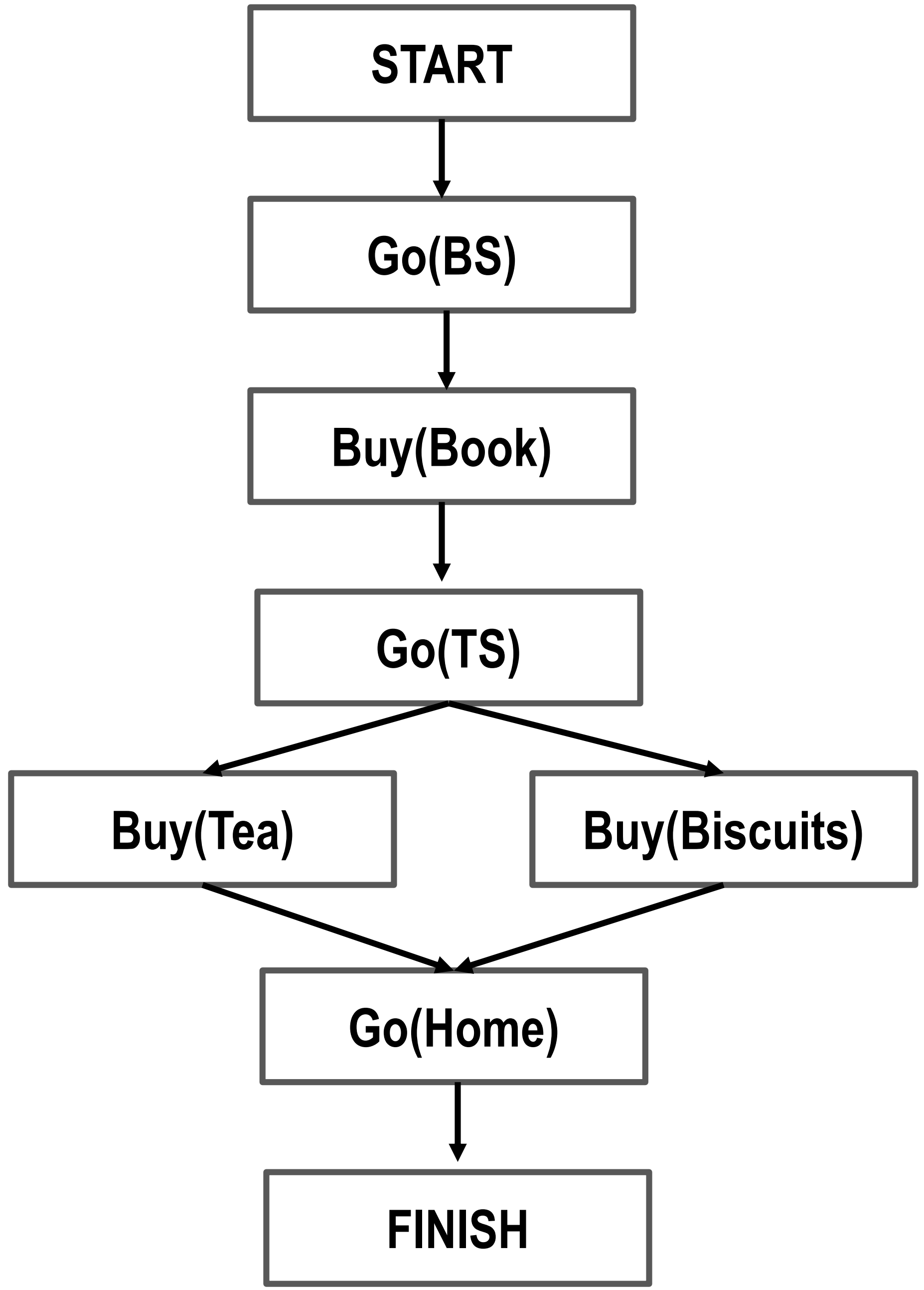


The red link prevents me from going to TS before buying the book


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

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 Algorithm (Contd.)

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

Proc 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 Algorithm (Contd.)

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, *¬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 possible 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

s.t. *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