Classical Planning

Vijay Kumar Meena
Assistant Professor
KIIT University

Acknowledgement

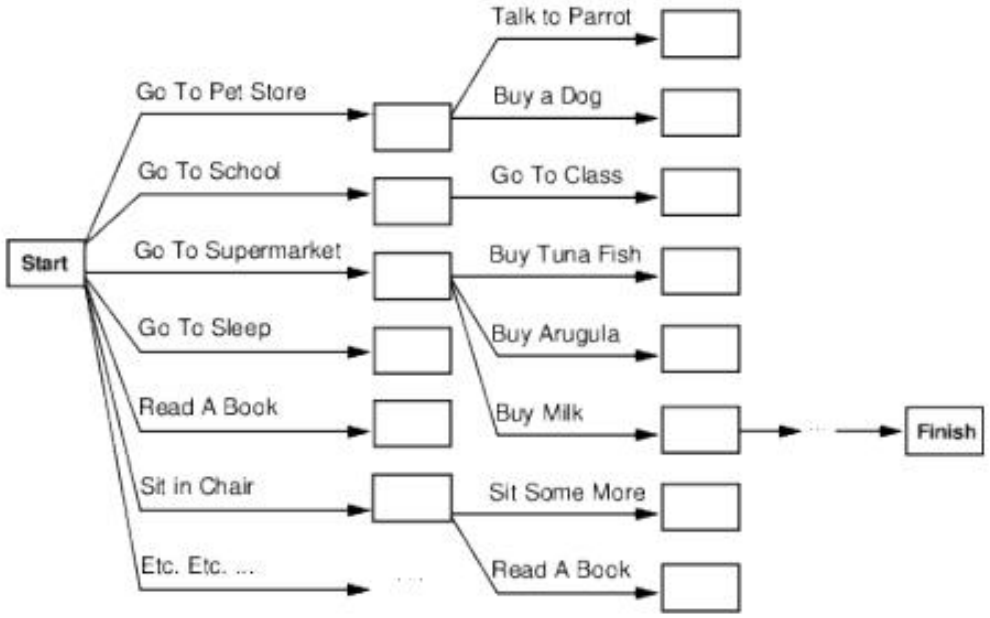
- These slides are taken (as it is or with some modifications) from various resources including -
  - Artificial Intelligence: A Modern Approach (AIMA) by Russell and Norvig.
  - Slides of Prof. Mausam (IITD)
  - Slides of Prof. Rohan Paul (IITD)
  - Slides of Prof. Brian Yu (Harvard University)

Planning

- We have defined AI as the study of rational action, which means that planning—devising a plan of action to achieve one's goals—is a critical part of AI.
- We have seen two examples of planning agents so far: the search-based problem solving agent and logical agent.
- The search-based problem solving agent can find sequences of actions that result in a goal state. But it deals with atomic representations of states an thus needs good domain-specific heuristic to perform well.
- The propositional logical agent can find plans without domain-specific heuristics because it uses domain-independent heuristics based on the logical structure of the problem. But it relies on ground (variable-free) propositional inference, which means that it may be swamped when there are many actions and states.
- For example, in the wumpus world, the simple action of moving a step forward had to be repeated for all four agent orientations, T time steps, and n^2 current locations.

Search vs. Planning

- Consider the task *get milk, bananas, and a cordless drill*
- Standard search algorithms seem to fail miserably



- Too many choices, no immediate feedback

Search vs. Planning

- Planning systems do the following
  1. improve action and goal representation to allow selection
  2. divide-and-conquer by **subgoaling**
  3. relax requirement for sequential construction of solutions

- Differences

	Search	**Planning**
States	Data structures	Logical sentences
Actions	Program code	Preconditions/outcomes
Goal	Program code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Planning

- In response to this, planning researchers have settled on a factored representation— one in which a state of the world is represented by a collection of variables.
- A language called **PDDL (**Planning Domain Definition Language) is used to express all $4Tn^2$ actions with one action schema.
- Let's see how PDDL describes the four things we need to define a search problem: the initial state, the actions that are available in a state, the result of applying an action, and the goal test.

PDDL

- Each **state** is represented as a conjunction of fluents that are ground, functionless atoms.
- For example, Poor ∧ Unknown might represent the state of a hapless agent, and a state in a package delivery problem might be At(Truck 1, Melbourne) ∧ At(Truck 2, Sydney).
- he following fluents are not allowed in a state: At(x, y) (because it is non-ground), ¬Poor (because it is a negation), and At(Father (Fred), Sydney) (because it uses a function symbol).

PDDL

- Actions are described by a set of action schemas that implicitly define the ACTIONS(s) and RESULT(s, a) functions needed to do a problem-solving search.
- A set of ground (variable-free) actions can be represented by a single action schema.
- For example, here is an action schema for flying a plane from one location to another -

$$Action(Fly(p, from, to),$$
$$\text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT:} \neg At(p, from) \wedge At(p, to))$$

$$Action(Fly(P_1, SFO, JFK),$$
$$\text{PRECOND:} At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$$
$$\text{EFFECT:} \neg At(P_1, SFO) \wedge At(P_1, JFK))$$

PDDL

- In PDDL the times and states are implicit in the action schemas: the precondition always refers to time t and the effect to time t + 1.
- A set of action schemas serves as a definition of a planning domain. A specific problem within the domain is defined with the addition of an initial state and a goal.

Planning Domain Definition Language (PDDL)

PDDL is derived from the STRIPS planning language.

STanford Research Institute Problem Solver

– Initial and goal states.
– A set of $\textsc{Actions}(s)$ in terms of preconditions and effects.
– Closed world assumption: Unmentioned state variables are assumed false.

Example:

$\textsc{Action}$: Fly($from$, to)
$\textsc{Precondition}$: At(p, $from$), Plane(p), Airport($from$), Airport(to)
$\textsc{Effect}$: ¬At(p, $from$), At(p, to)

PDDL/STRIPS operators

Tidily arranged actions descriptions, restricted language

ACTION: Buy(x)
PRECONDITION: At(p), Sells(p, x)
EFFECT: Have(x)

[Note: this abstracts away many
important details of buying!]

Restricted language $\Rightarrow$ efficient algorithm
Precondition: conjunction of positive literals
Effect: conjunction of literals

A complete set of STRIPS operators can be translated
into a set of successor-state axioms

Example: Air Cargo Transport

- Air cargo transport problem involves loading and unloading cargo and flying it from place to place.
- The problem can be defined with three actions: Load, Unload, and Fly.
- The actions affect two predicates: In(c, p) means that cargo c is inside plane p, and At(x, a) means that object x (either plane or cargo) is at airport a. Note that some care must be taken to make sure the At predicates are maintained properly. When a plane flies from one airport to another, all the cargo inside the plane goes with it.
- The following plan is a solution to the problem -

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$$

Example: Air Cargo Transport

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

Example: The Spare Tire Problem

- Consider the problem of changing a flat tire, The goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk.
- To keep it simple, our version of the problem is an abstract one, with no sticky lug nuts or other complications.
- There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight.
- We assume that the car is parked in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.
- A solution to the problem is **[Remove(Flat, Axle), Remove(Spare, Trunk),PutOn(Spare, Axle)]**.

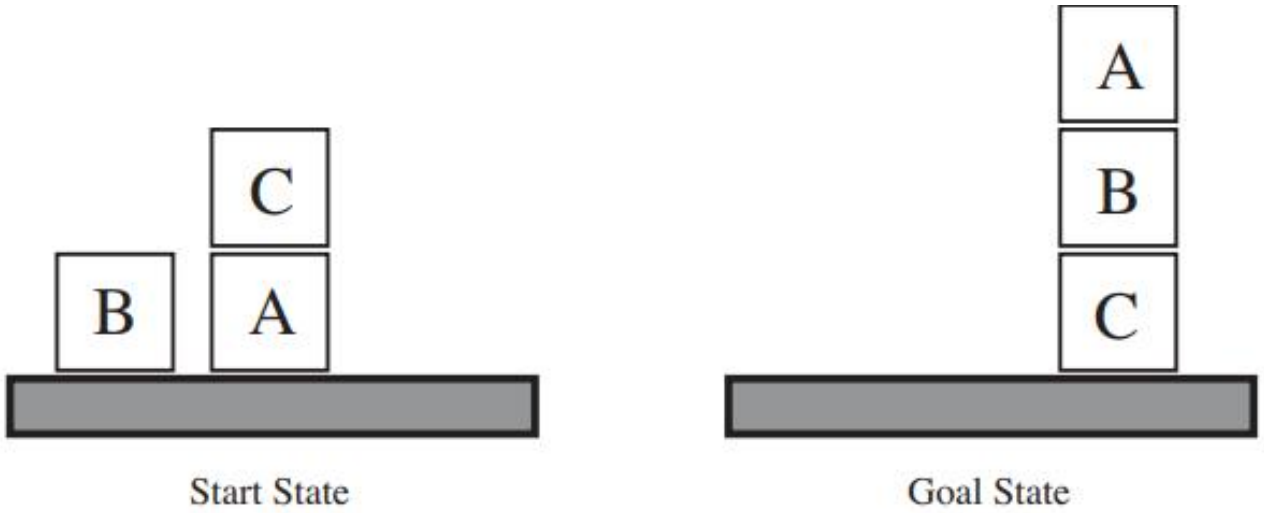
Example: The Spare Tire Problem

$$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$$
$$Goal(At(Spare, Axle))$$
$$Action(Remove(obj, loc),$$
$$\quad \text{PRECOND: } At(obj, loc)$$
$$\quad \text{EFFECT: } \neg At(obj, loc) \land At(obj, Ground))$$
$$Action(PutOn(t, Axle),$$
$$\quad \text{PRECOND: } Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$$
$$\quad \text{EFFECT: } \neg At(t, Ground) \land At(t, Axle))$$
$$Action(LeaveOvernight,$$
$$\quad \text{PRECOND:}$$
$$\quad \text{EFFECT: } \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$$
$$\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$$

Figure 10.2 The simple spare tire problem.

Example: The Blocks World

- One of the most famous planning domains is known as the blocks world. This domain consists of a set of cube-shaped blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another.
- The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it.



Example: The Block World

- We use On(b, x) to indicate that block b is on x, where x is either another block or the table.
- The action for moving block b from the top of x to the top of y will be Move(b, x, y).
- Now, one of the preconditions on moving b is that no other block be on it. In first-order logic, this would be $\neg \exists$ x On(x, b) or, alternatively, $\forall$ x $\neg$On(x, b). Basic PDDL does not allow quantifiers, so instead we introduce a predicate Clear (x) that is true when nothing is on x.

Example: The Block World Problem

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

Figure 10.3 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

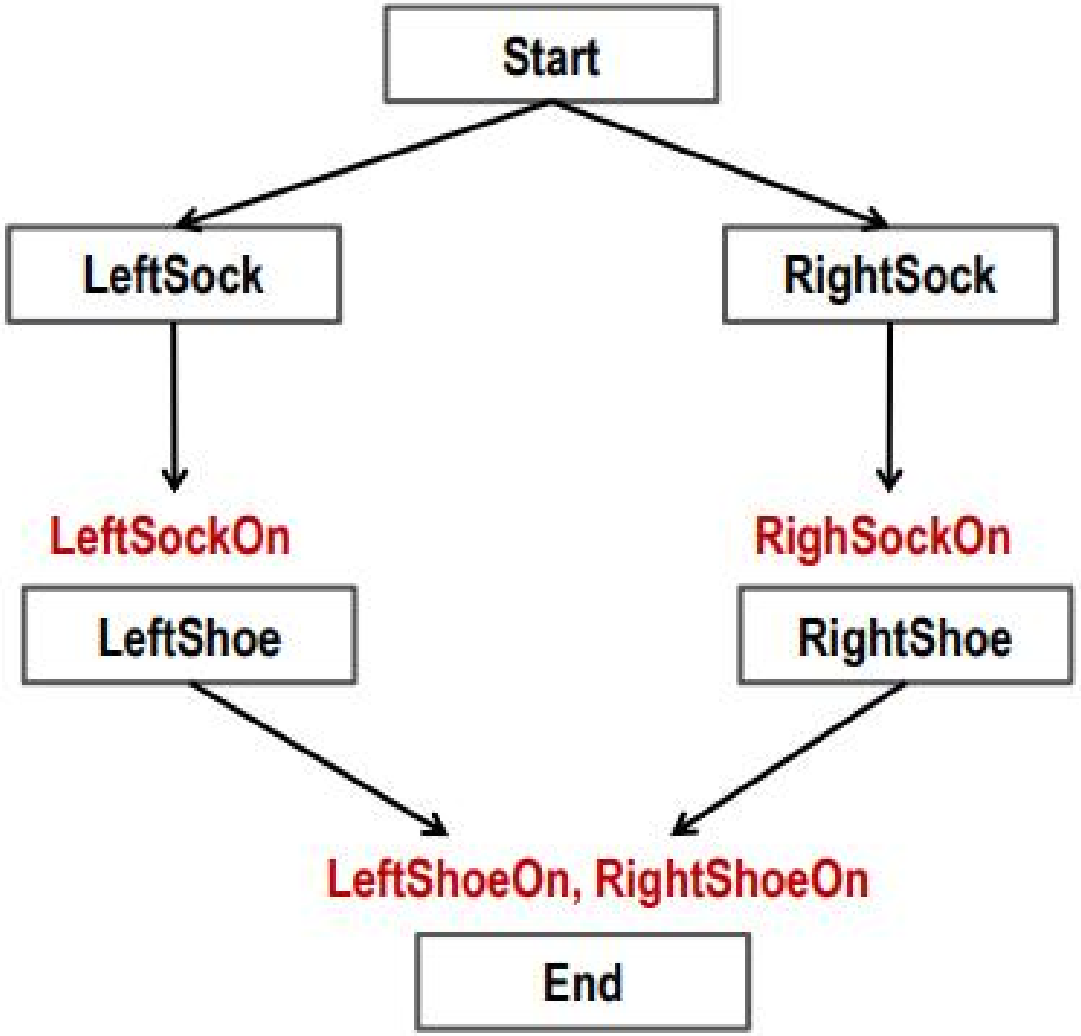
Partial Order Planning

- The solution of planning problems we have seen so far are total order, a strict linear sequence of actions with fix order.
- Partial Order Planning (POP) is a type of AI planning where actions are only partially ordered rather than strictly sequential. This means that the planner determines the necessary steps to achieve a goal but does not fully specify their order unless required by dependencies.
- Partially ordered plans are created by a search through the space of plans rather than through the state space.
- We start with the empty plan consisting of just the initial state and the goal, with no actions in between. The search procedure then looks for a flaw in the plan, and makes an addition to the plan to correct the flaw (or if no correction can be made, the search backtracks and tries something else). A flaw is anything that keeps the partial plan from being a solution.

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"

Partial Order Planning



Partial Order Planning

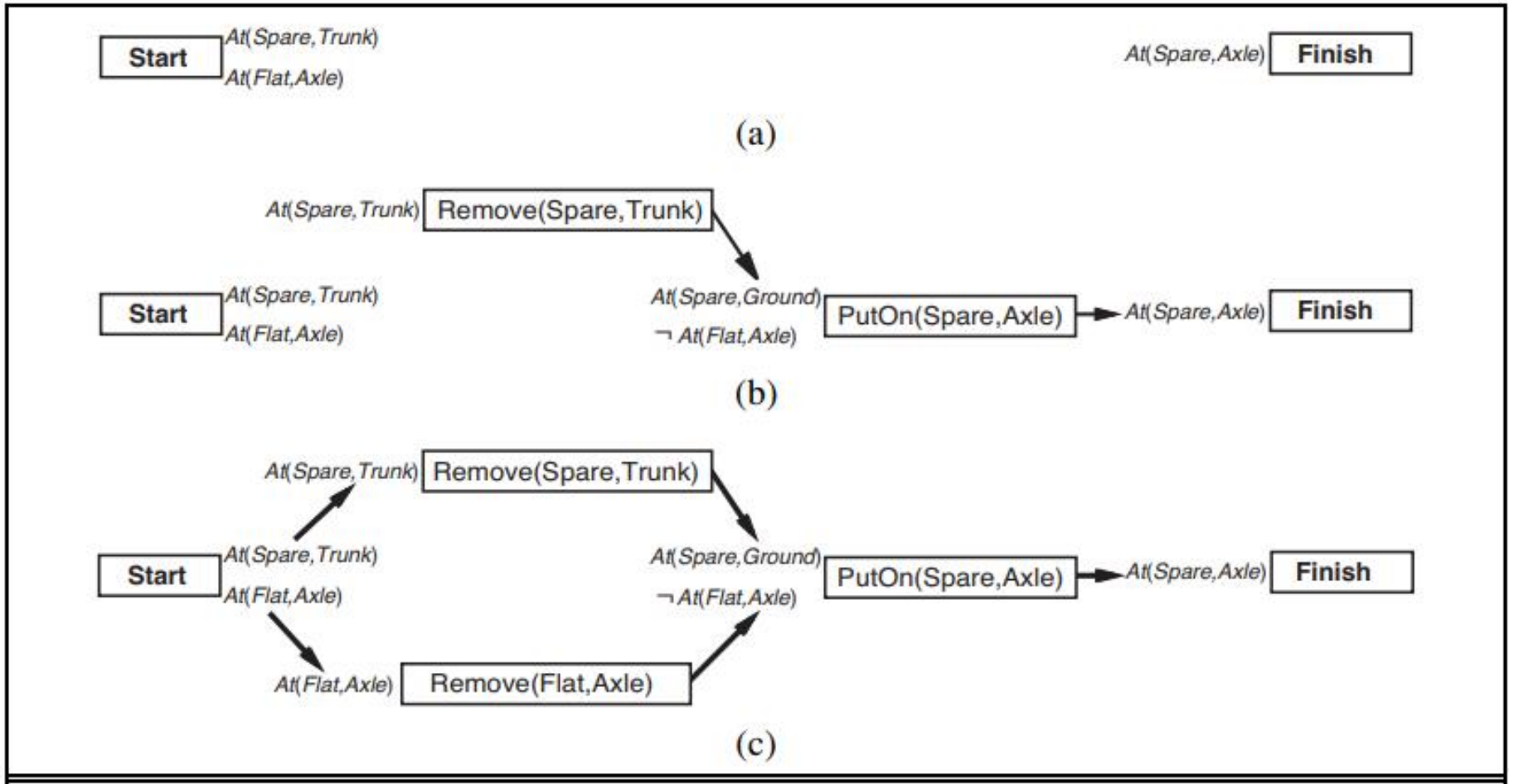


Figure 10.13 (a) the tire problem expressed as an empty plan. (b) an incomplete partially ordered plan for the tire problem. Boxes represent actions and arrows indicate that one action must occur before another. (c) a complete partially-ordered solution.

Example: Get Tea, Biscuits, Book

Representing States

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

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

- 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) ∧ Path(here, there)
EFFECT: At(there) ∧ ¬At(here))

POP Example: Get Tea, Biscuits, Book

Initial state:

Op(ACTION: Start,
EFFECT: At(Home) ∧ Sells(BS, Book)
∧ Sells(TS, Tea)
∧ Sells(TS, Biscuits))

Goal state:

Op(ACTION: Finish,
PRECOND: At(Home) ∧ Have(Tea)
∧ Have(Biscuits)
∧ Have(Book))

Actions:

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

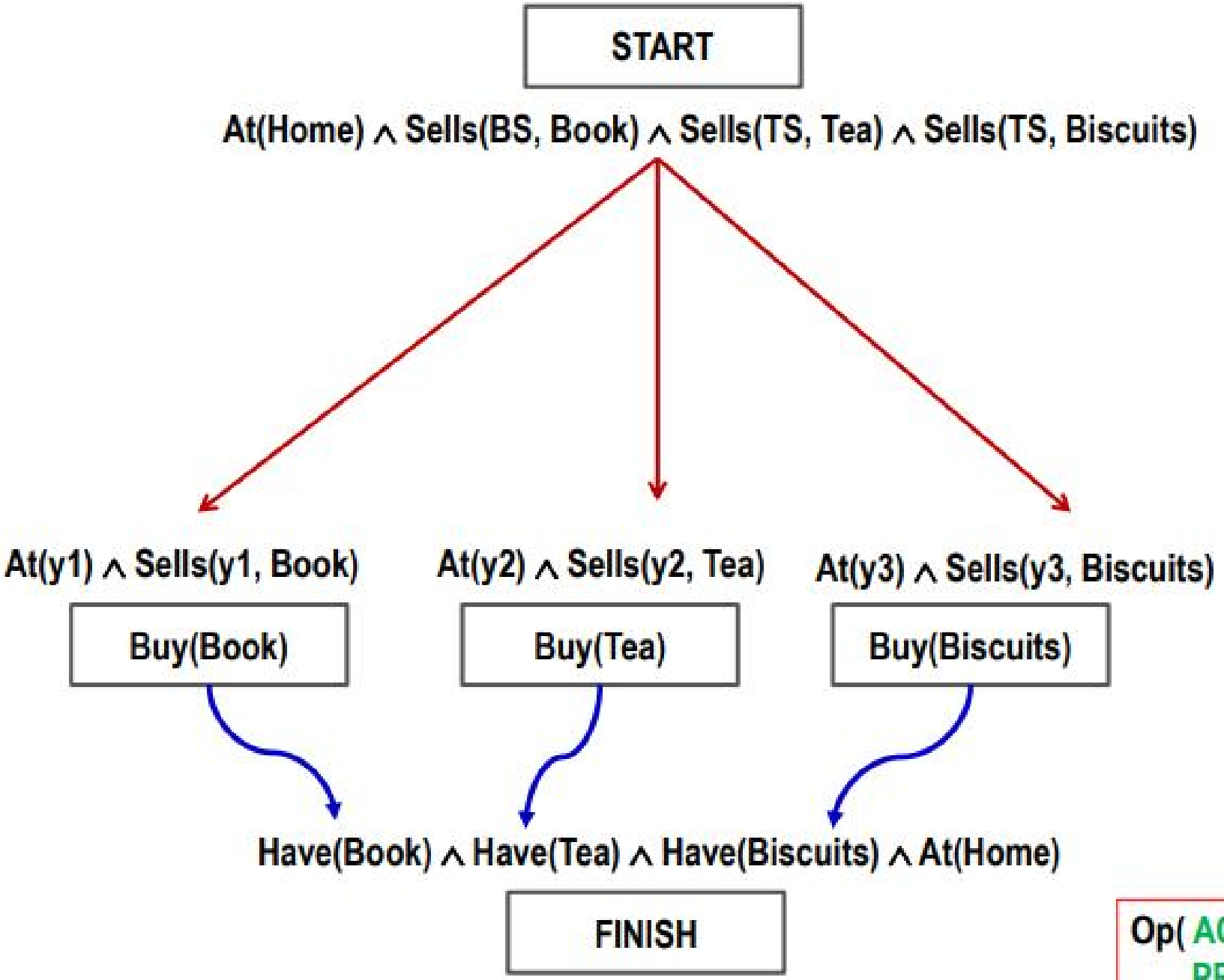
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))

START

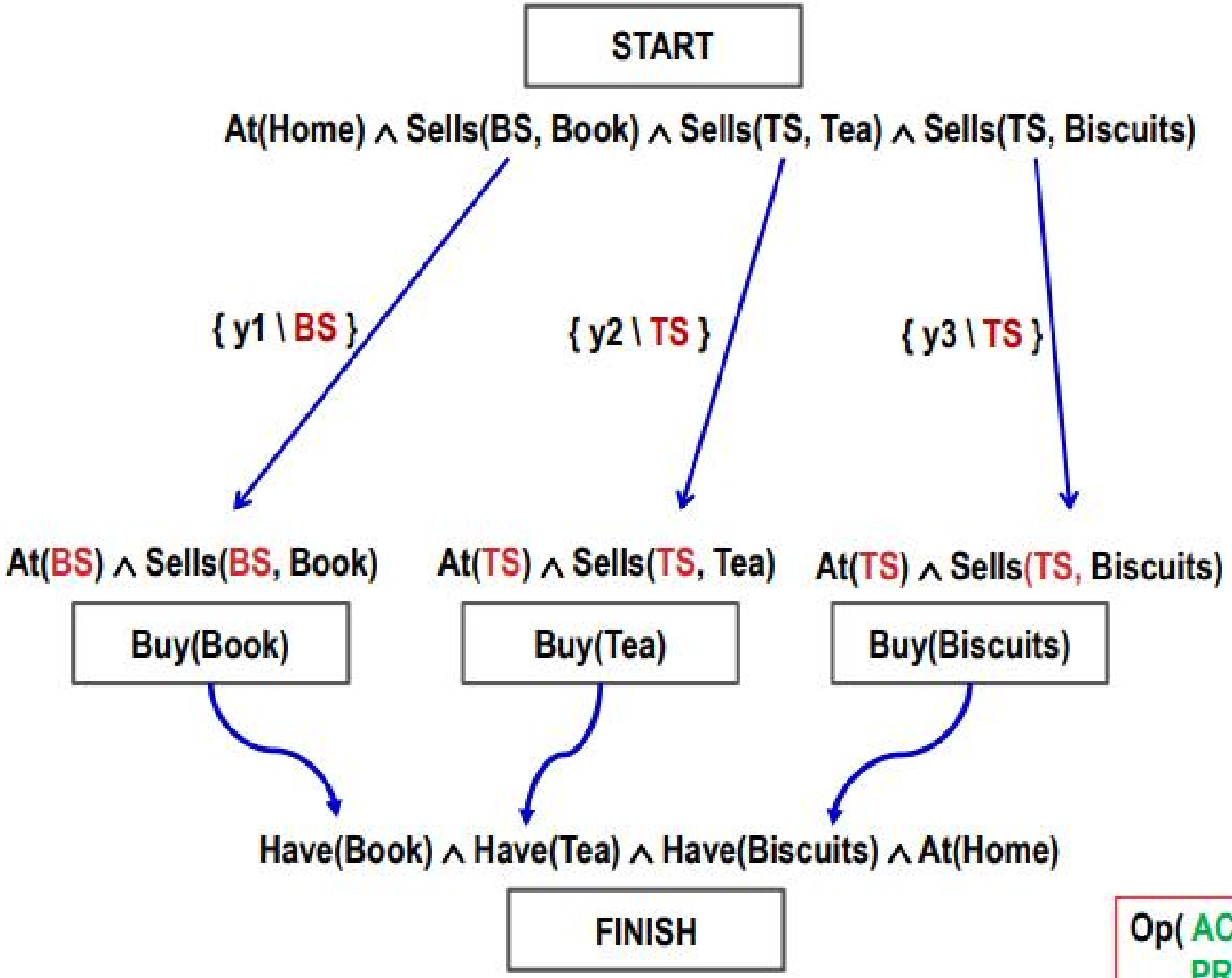
$At(Home) \wedge Sells(BS, Book) \wedge Sells(TS, Tea) \wedge Sells(TS, Biscuits)$

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

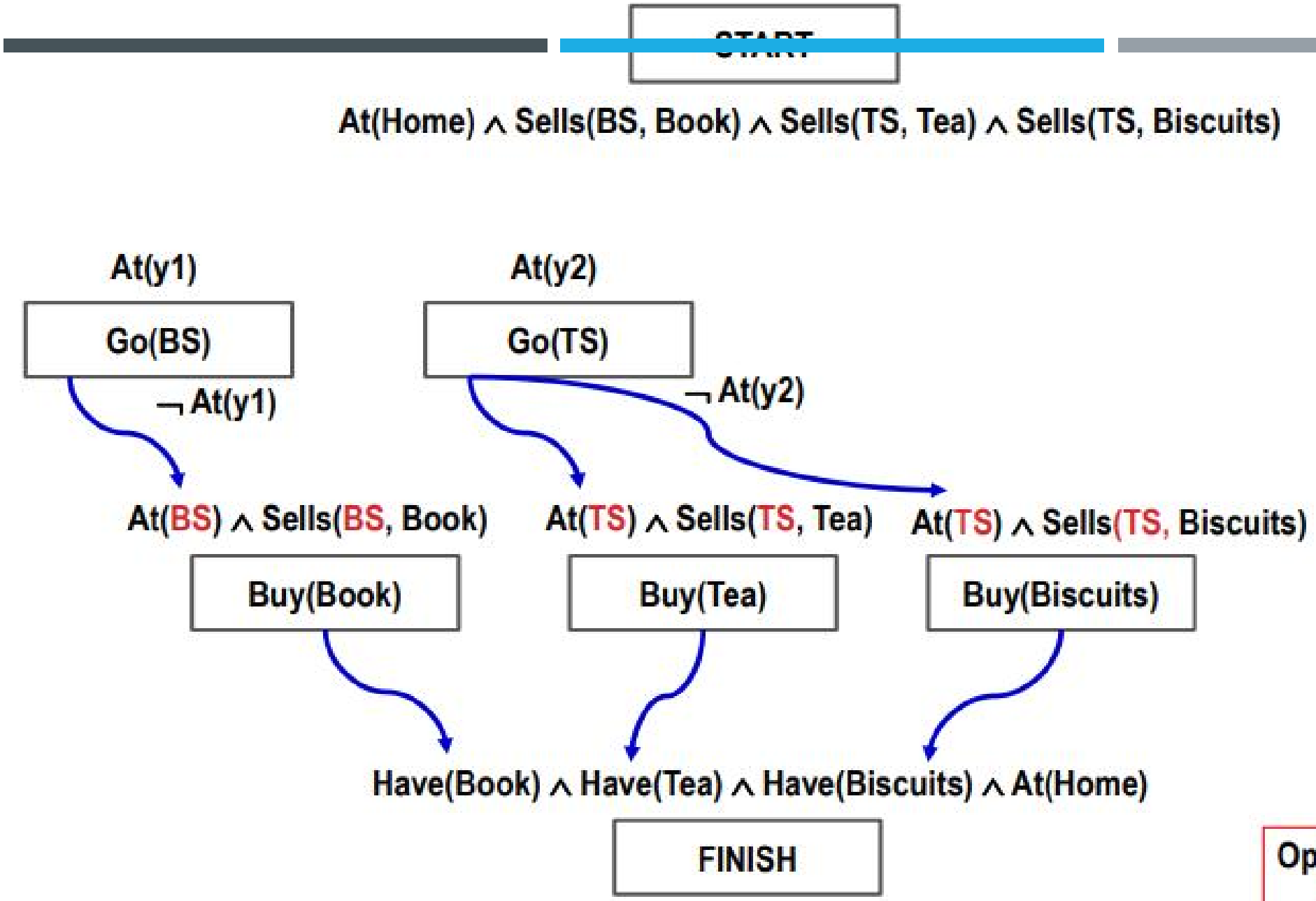
FINISH



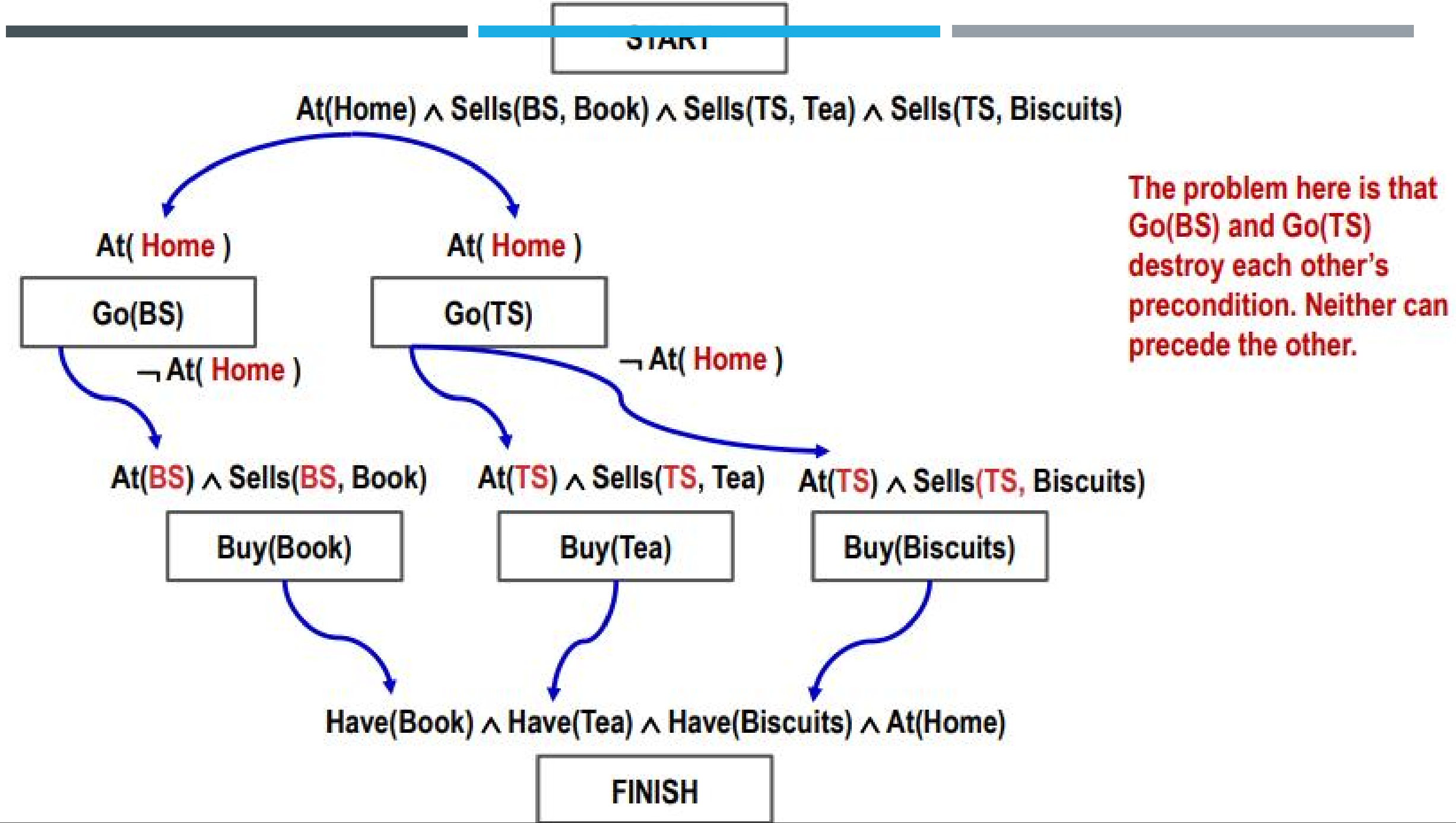
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))

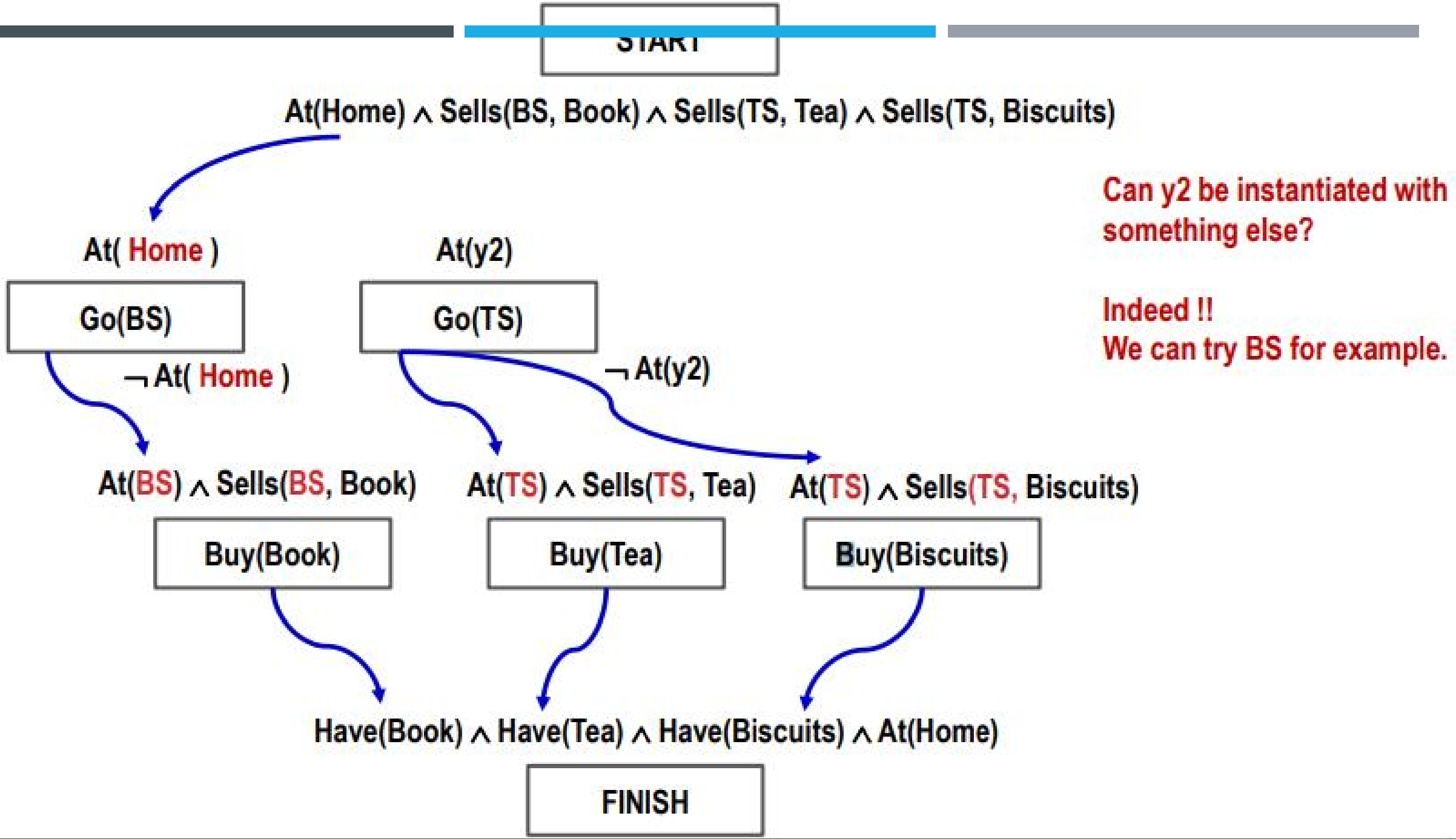


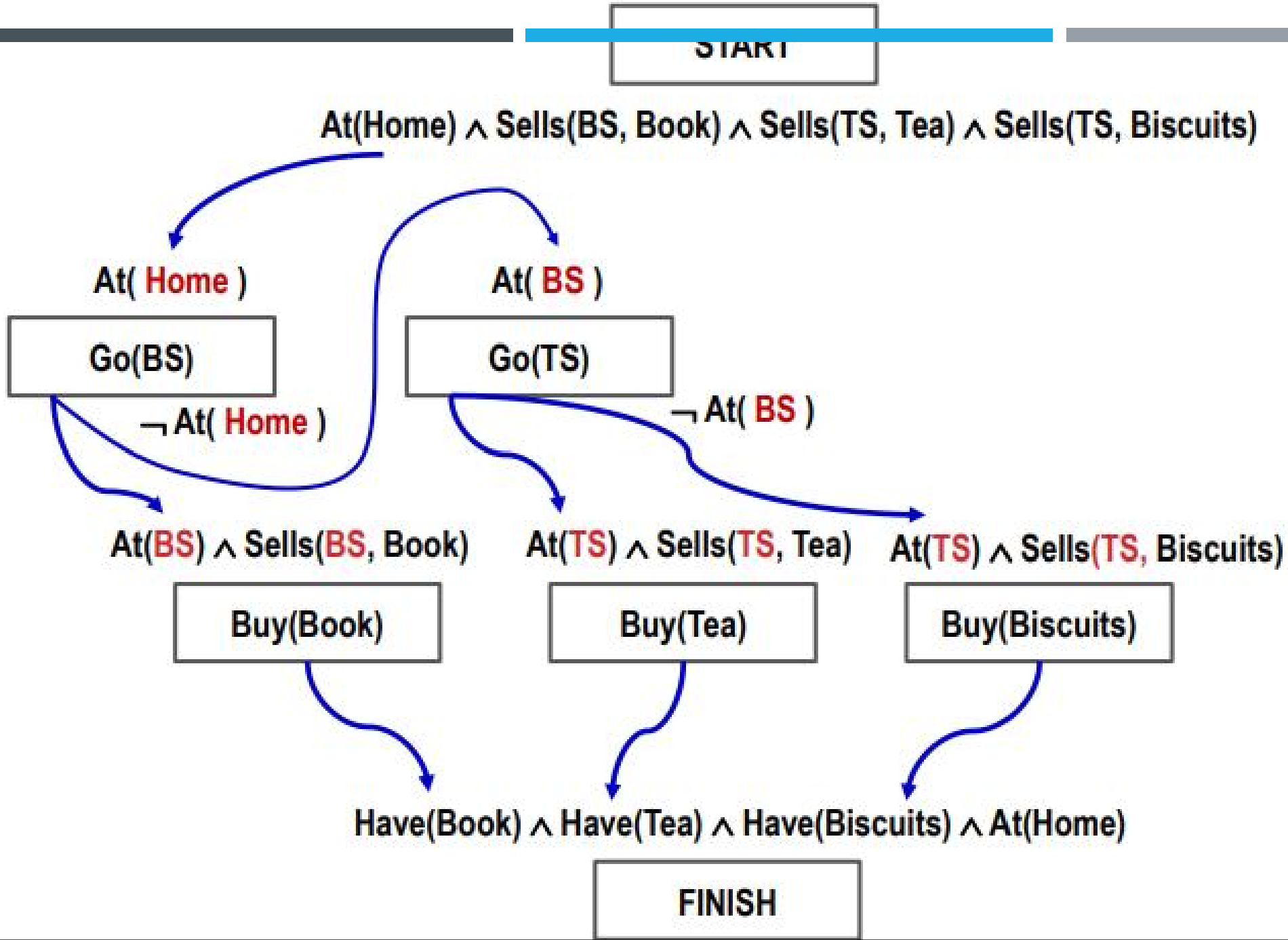
Op(ACTION: Buy(x),
PRECOND: At(y) ∧ Sells(y, x),
EFFECT: Have(x))

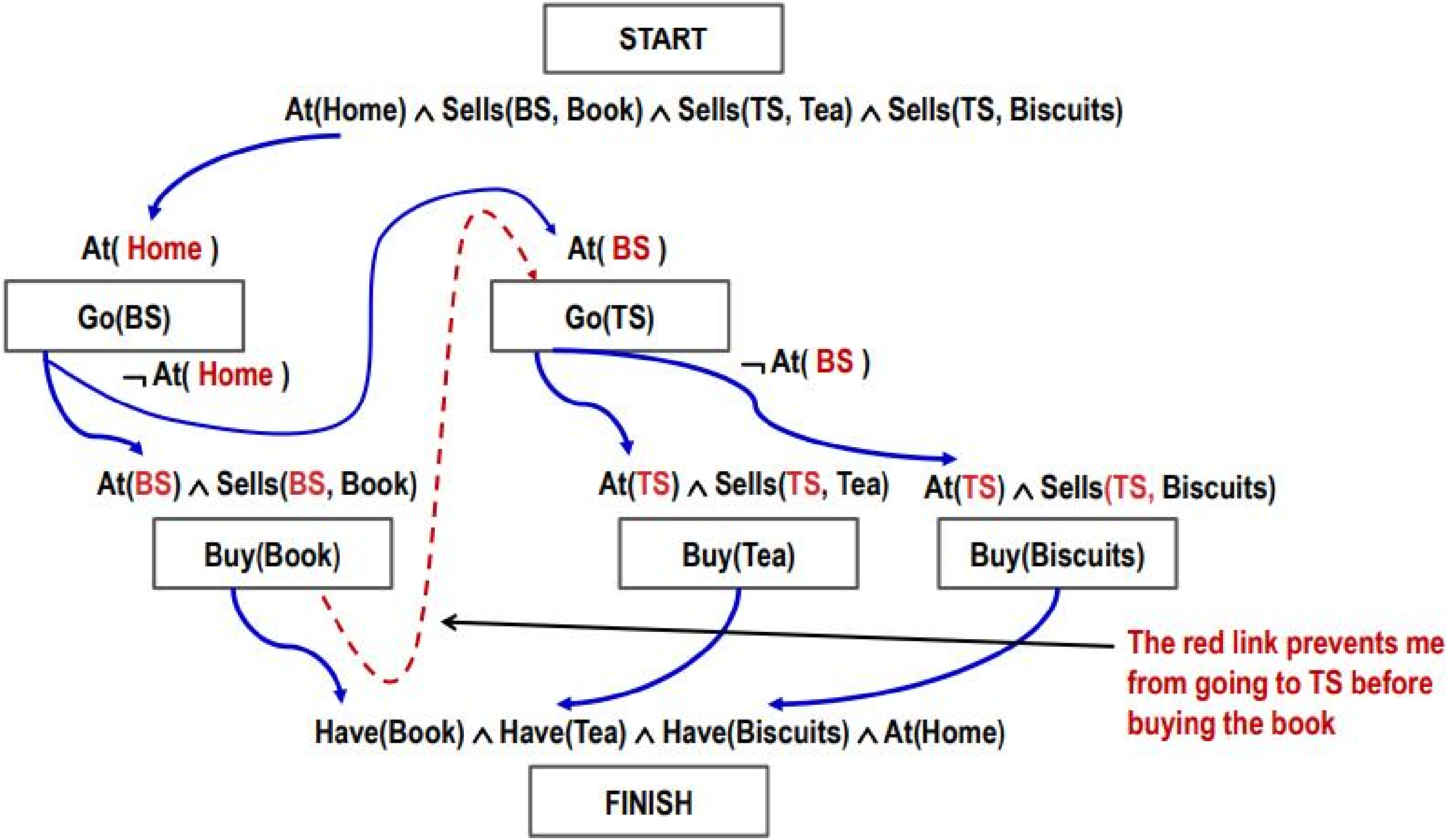


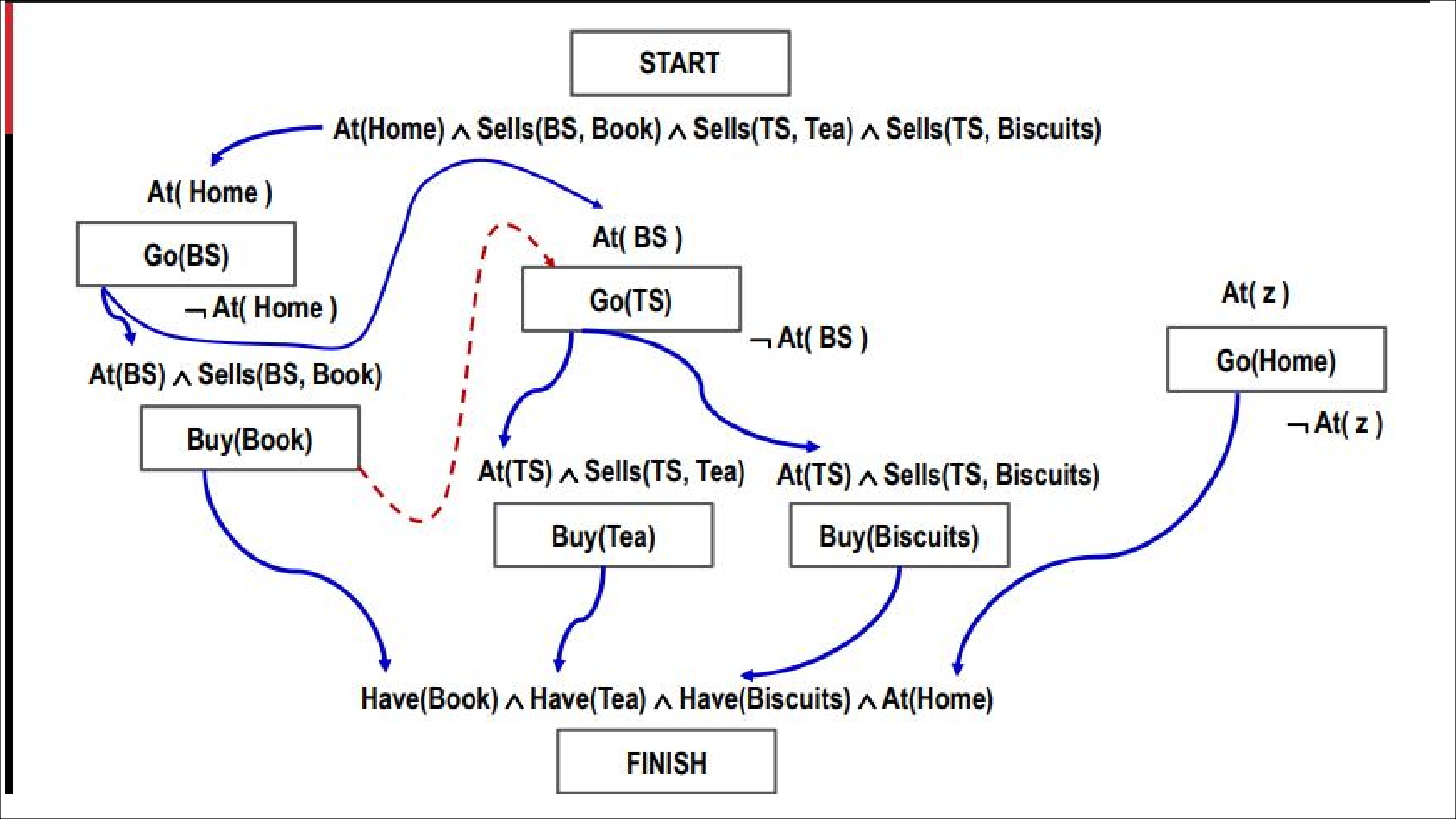
Op(ACTION: Go(y),
PRECOND: At(x),
EFFECT: At(y) ∧ ¬At(x))

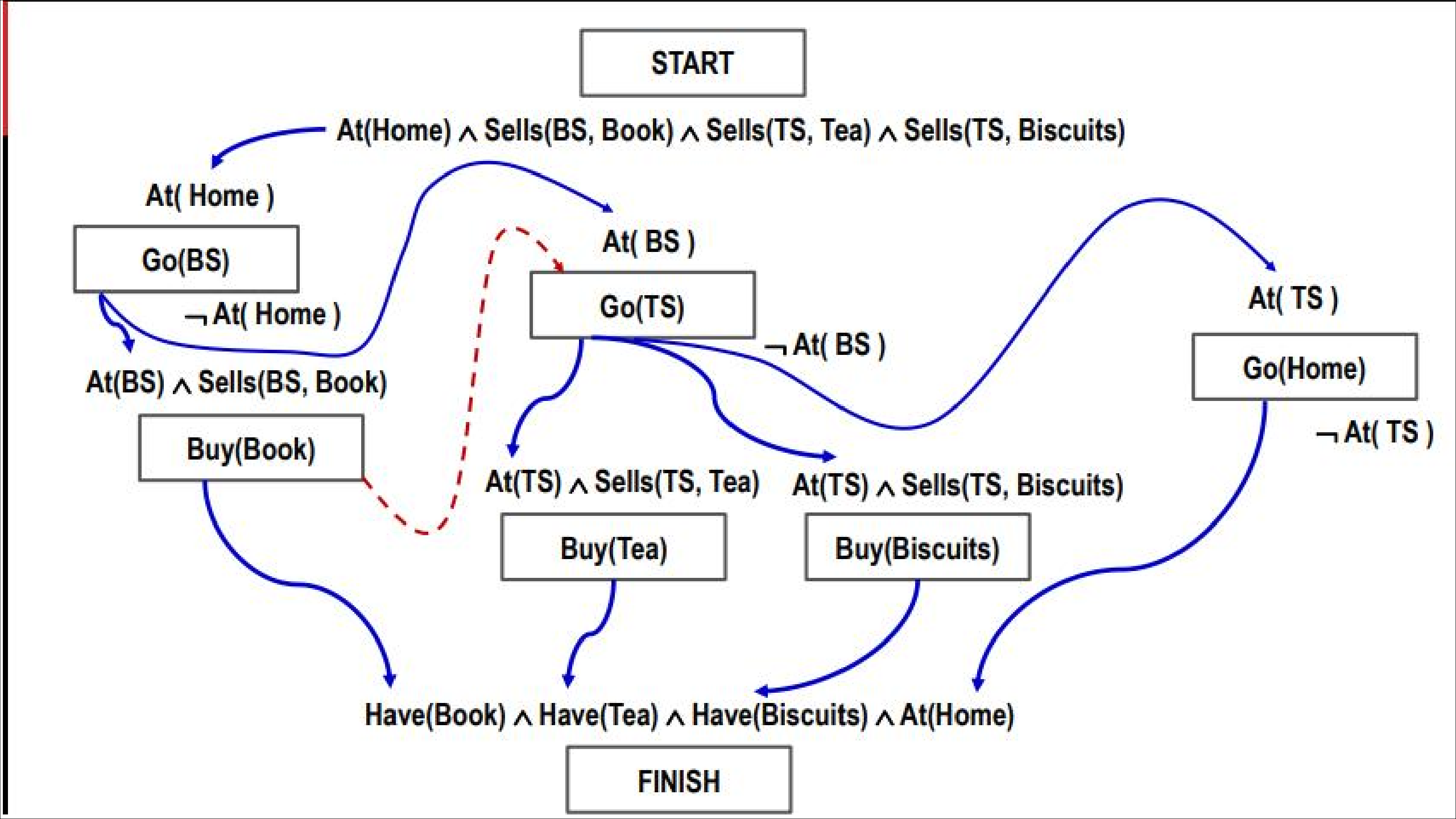

START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At(Home)
At(Home)
Go(BS)
Go(TS)
¬ At(Home)
¬ At(Home)
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH
The problem here is that Go(BS) and Go(TS) destroy each other's precondition. Neither can precede the other.

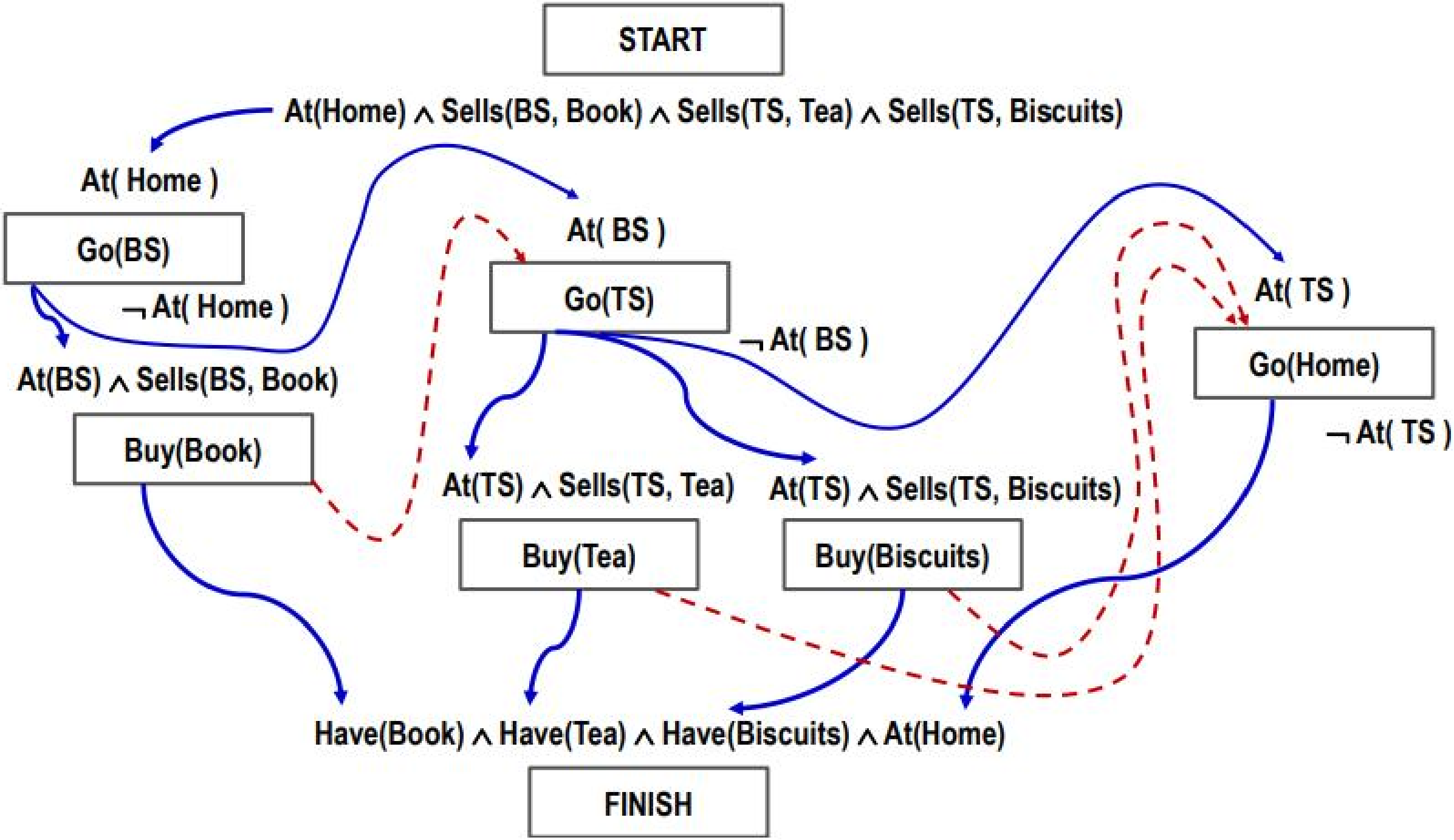

START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At(Home)
At(y2)
Go(BS)
Go(TS)
¬ At(Home)
¬ At(y2)
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH
Can y2 be instantiated with something else?
Indeed !!
We can try BS for example.

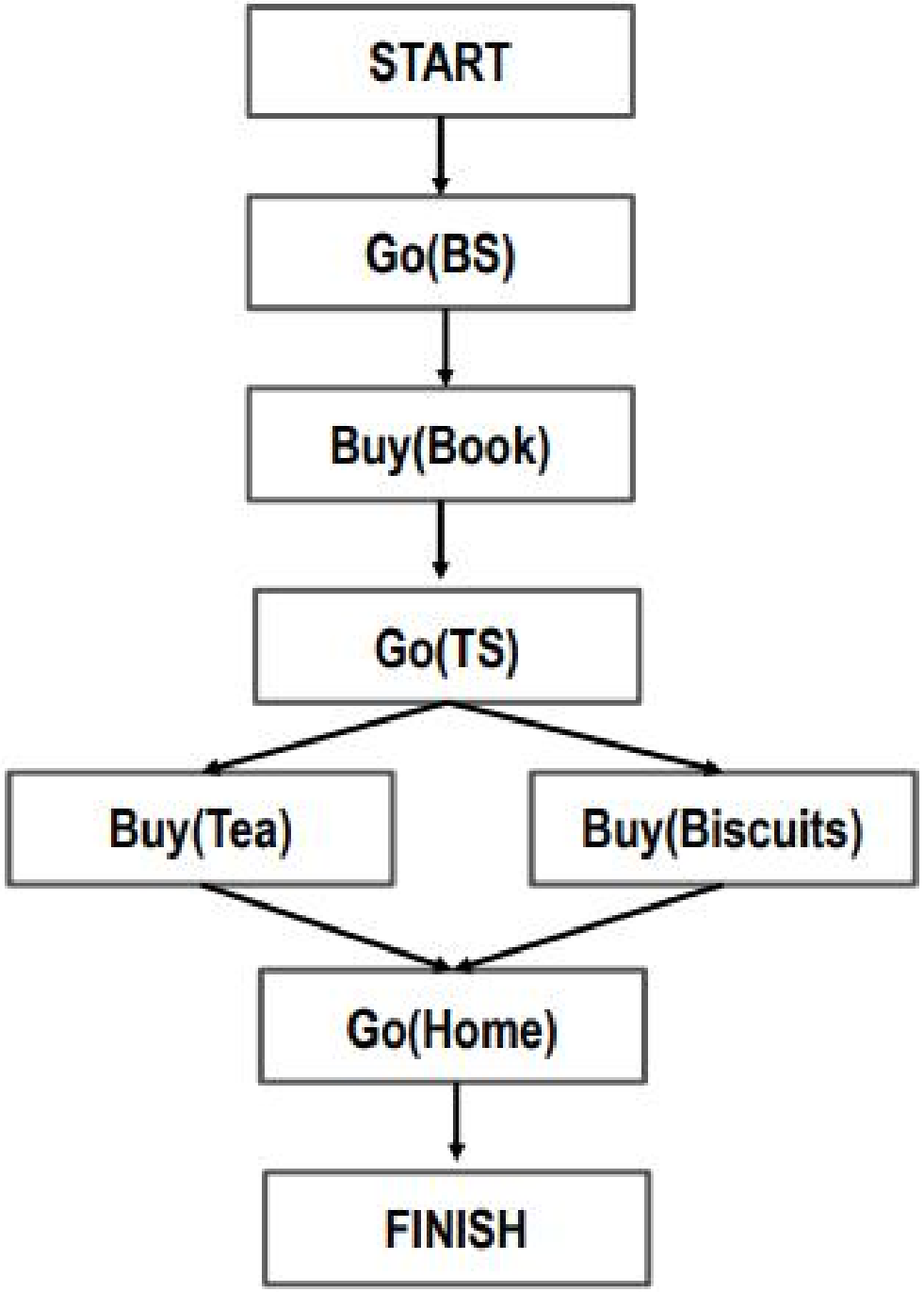

START
At(Home) ∧ Sells(BS, Book) ∧ Sells(TS, Tea) ∧ Sells(TS, Biscuits)
At(Home)
At(BS)
Go(BS)
Go(TS)
¬ At(Home)
¬ At(BS)
At(BS) ∧ Sells(BS, Book)
At(TS) ∧ Sells(TS, Tea)
At(TS) ∧ Sells(TS, Biscuits)
Buy(Book)
Buy(Tea)
Buy(Biscuits)
Have(Book) ∧ Have(Tea) ∧ Have(Biscuits) ∧ At(Home)
FINISH


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