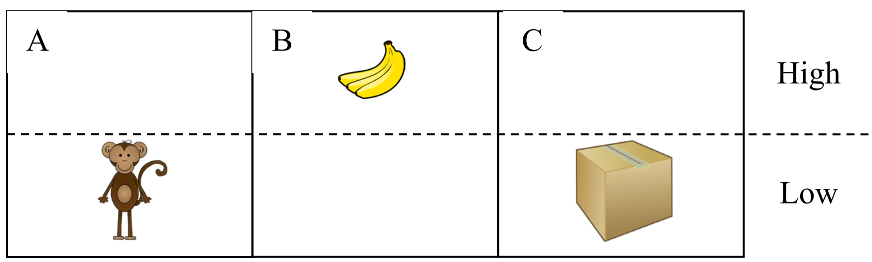
COS30019 - Introduction to Artificial Intelligence
Tutorial Problems Week 9

Task 1: The monkey-and-bananas problem is faced by a monkey in a laboratory with some bananas hanging out of reach from the ceiling. 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, *ClimbUp* onto or *ClimbDown* from an object, and *Grasp* or *Ungrasp* an object. Grasping results in holding the object if the monkey and object are in the same place at the same height. If Monkey has bananas, it cannot push the box.



1. Write down the initial state description.

   Initial: MonkeyAt(A) ∧ At(Bananas, B) ∧ At(Box, C) ∧ CanPush ∧ MonkeyHeight(Low) ∧ AtHeight(Box, Low) ∧ AtHeight(Bananas, High)

QUESTION: Why don't we write **At(Monkey, A)** and **AtHeight(Monkey, Low)?**

2. Write down STRIPS-style definitions of the six actions.

   <u>Go from one place to another:</u>
   *Action(***Go(start, end),**
   PRECOND: MonkeyAt(start)
   EFFECT: ¬MonkeyAt(start) ∧ MonkeyAt(end))

   <u>Push an object from one place to another (assuming that the object has to be at height *Low*):</u>
   *Action(***Push(obj, start, end),**
   PRECOND: MonkeyAt(start) ∧ At(obj, start) ∧ CanPush ∧ MonkeyHeight(Low) ∧ AtHeight(obj, Low)
   EFFECT: ¬MonkeyAt(start) ∧ MonkeyAt(end) ∧ ¬At(obj, start) ∧ At(obj, end))

ClimbUp onto an object:

*Action(***ClimbUp(obj, location),**

PRECOND: MonkeyAt(location) ∧ At(obj, location) ∧ MonkeyHeight(Low) ∧ AtHeight(obj, Low)

EFFECT: ¬MonkeyHeight(Low) ∧ MonkeyHeight(High))

ClimbDown from an object:

*Action(***ClimbDown(obj, location),**

PRECOND: MonkeyAt(location) ∧ At(obj, location) ∧ MonkeyHeight(High) ∧ AtHeight(obj, Low)

EFFECT: ¬MonkeyHeight(High) ∧ MonkeyHeight(Low))

Grasp an object:

*Action(***Grasp(obj, location, height),**

PRECOND: MonkeyAt(location) ∧ At(obj, location) ∧ MonkeyHeight(height) ∧ AtHeight(obj, height)

EFFECT: MonkeyHas(obj) ∧ ¬CanPush) ∧ ¬AtHeight(obj, height) ∧ ¬At(obj, location))

UnGrasp an object:

*Action(***UnGrasp(obj, location, height),**

PRECOND: MonkeyAt(location) ∧ MonkeyHeight(height) ∧ MonkeyHas(obj)

EFFECT: ¬MonkeyHas(obj) ∧ CanPush) ∧ AtHeight(obj, height) ∧ At(obj, location))

3. Suppose the monkey wants to fool the scientists, who are off to tea, by grabbing the bananas, but leaving the box in its original place. Can this goal be written and solved by a STRIPS-style system?

   A specific goal (for the above scenario) is:
   Goal: **At(Box, C) ∧ AtHeight(Box, Low) ∧ MonkeyHas(Bananas)**
   (the box is in in its original place, only the bananas are with monkey)

   A general goal for this would require the agent (i.e. the monkey) to use a variable to store the box's original location (based on the agent's percepts) and the goal will include a sentence requiring the box to be back at that original place.

   [*This generic problem cannot be solved because the START has to be made of grounded literals only (so no variables).]*

Task 2:

1. Extend your wumpus world description of tutorial 7 with STRIPS descriptions for the operators forward (go one square in the current direction) and left (turn 90 degrees left, staying in the same square).

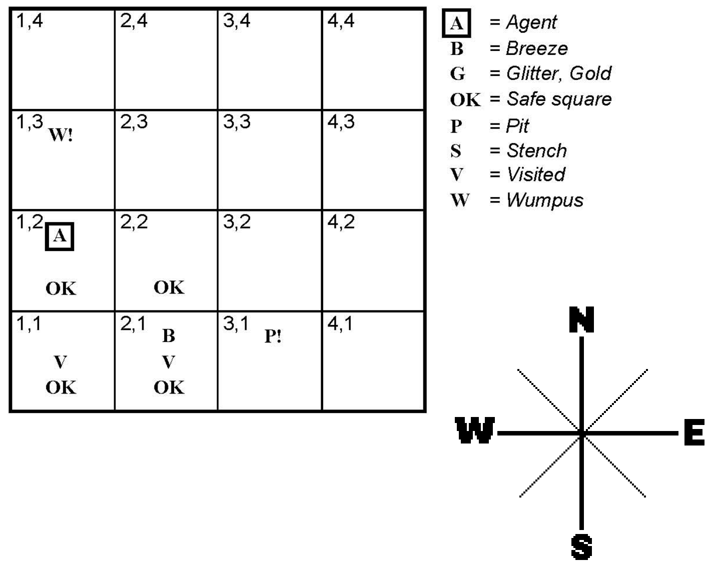
   We assume the following are included in the agent's knowledge base (in addition to other propositions about other domain knowledge such as breeze and pits, stench and wumpus, glitter and gold, etc.)

   LeftOf(North, West)
   LeftOf(West, South)
   LeftOf(South, East)
   LeftOf(East, North)

   NextTo([1,1], East, [2,1])
   NextTo([1,1], North, [1,2])

   NextTo([2,1], East, [3,1])
   NextTo([2,1], West, [1,1])
   NextTo([2,1], North, [2,2])

   NextTo([1,2], East, [2,2])
   NextTo([1,2], North, [1,3])
   NextTo([1,2], South, [1,1])
   …



   Actions
   // Agent currently in cell **c** and facing direction **d** (N,W,E,S) goes forward to cell **e**
   *Action(***Forward(c, d, e),**
   PRECOND: AgentAt(c) ∧ Facing(d) ⋀ NextTo(c, d, e)
   EFFECT: ¬At(Agent, c) ⋀ At(Agent, e))

   // Agent currently facing direction **d1** turns left to direction **d2**
   *Action(***Left(d1, d2),**
   PRECOND: Facing(d1) ⋀ LeftOf(d1, d2)
   EFFECT: ¬ Facing(d1) ⋀ Facing(d2))

2. Give an outline of how to find a plan for getting the gold.
   We assume that the agent is using progression planning from the current state (say, AgenAt([1,1]) ⋀ Facing(East)).
   The agent tries to apply a number of actions (including left and forward).
   When trying to apply forward, the agent will have to establish IsSafe(e) by querying the knowledge base (e is adjacent cell the agent is trying to move into).
   If the inference engine say YES, then forward() can be applied.
   The agent then continues to interleave between:
   - Executing action (exploring the environment)
   - Doing planning (determine what action to do next)

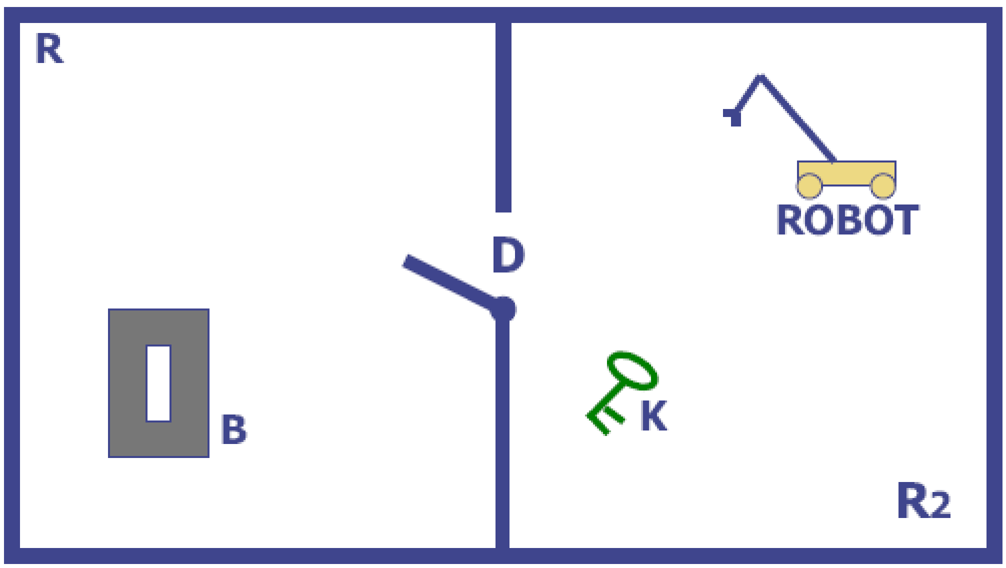
   until it reaches the cell with GOLD.

Task 3: A robot ROBOT operates in an environment made of two rooms R1 and R2 connected by a door D. A box B is located in R1 and the door's key is initially in R2. The door can be open or closed (and locked). The figure illustrates the initial state described by:

IN(ROBOT,R2)
IN(K,R2)
OPEN(D)

The actions are:

Grasp-Key-In-R2
Lock-Door
Go-From-R2-To-R1-With-Key
Put-Key-In-Box



Defined as follows:

Grasp-Key-In-R2
- P: IN(ROBOT,R2), IN(K,R2)
- E: HOLDING(ROBOT,K)

Lock-Door
- P: HOLDING(ROBOT,K), OPEN(D)
- E: ¬OPEN(D), LOCKED(D)

Go-From-R2-To-R1-With-Key
- P: IN(ROBOT,R2), HOLDING(ROBOT,K), OPEN(D)
- E: ¬IN(ROBOT,R2), ¬IN(K,R2), IN(ROBOT,R1), IN(K,R1)

Put-Key-In-Box
- P: IN(ROBOT,R1), HOLDING(ROBOT,K)
- E: ¬HOLDING(ROBOT,K), ¬IN(K,R1), IN(K,BOX)

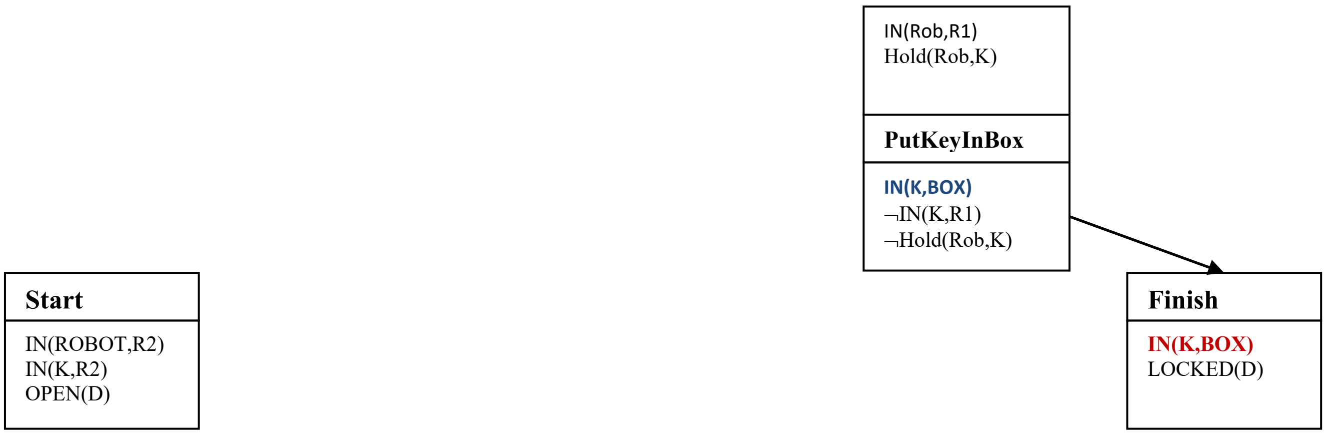
The goal is:
- IN(K,BOX), LOCKED(D)

Construct a partial order plan to solve this problem. Clearly indicate at each step the modifications made to the plan: the action added, the causal links added and/or the ordering constraints added. Indicate any threats at each step.

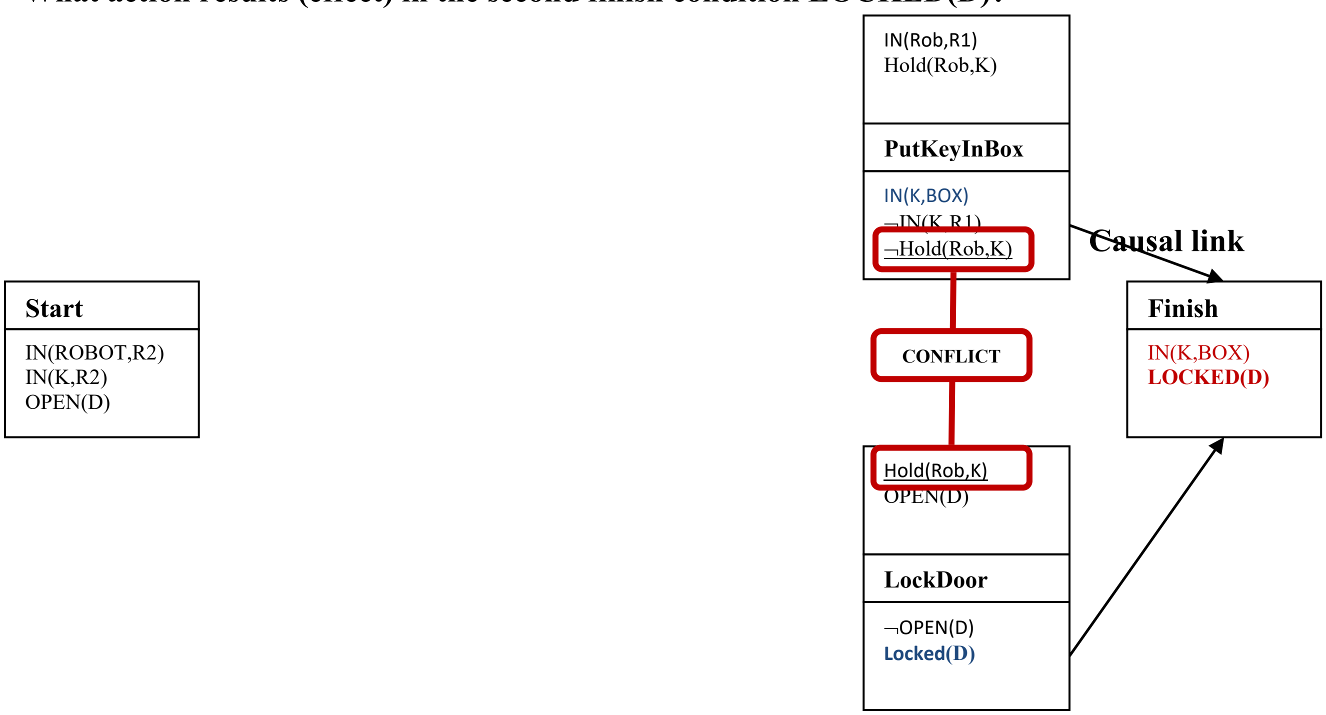
Causal link – categorically order actions (define general order as Action 2 must come somewhere after Action 1, but after Action 3).

1. **Define Start and Finish**

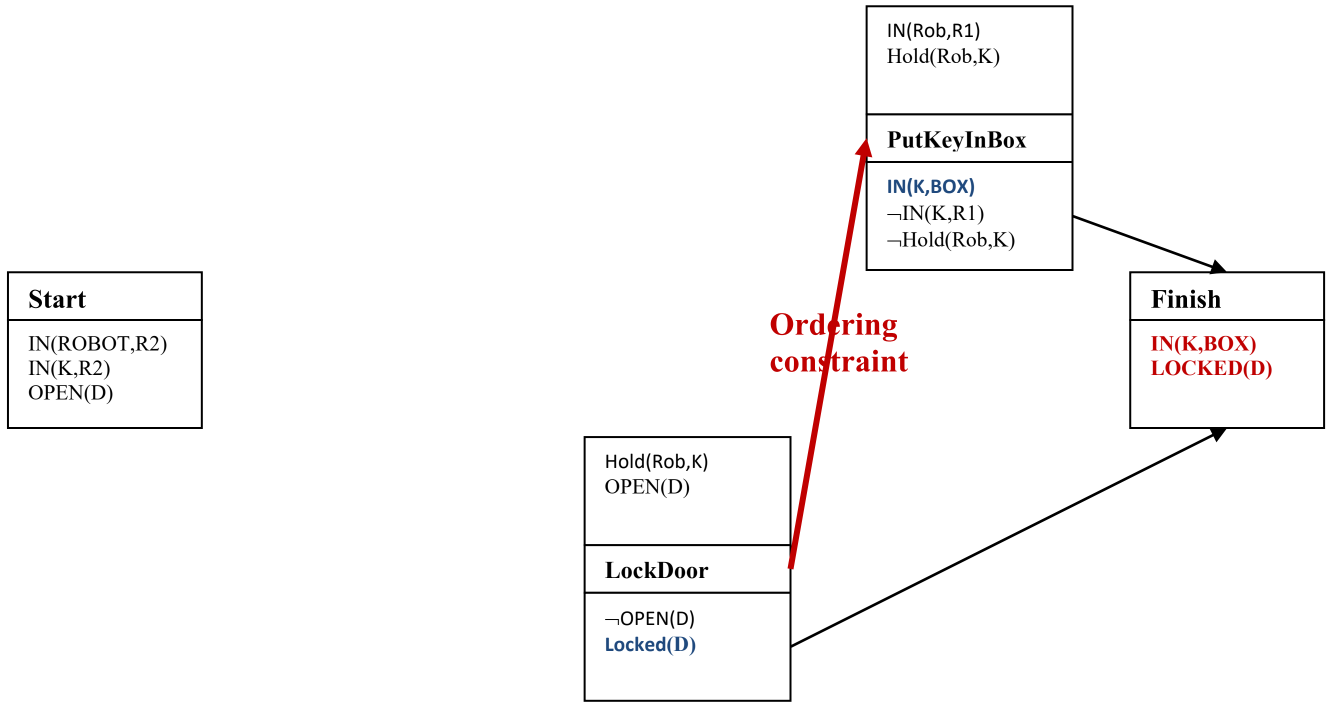
2. **What action results (effect) in the finish condition IN(K,BOX)?**



3. **What action results (effect) in the second finish condition LOCKED(D)?**

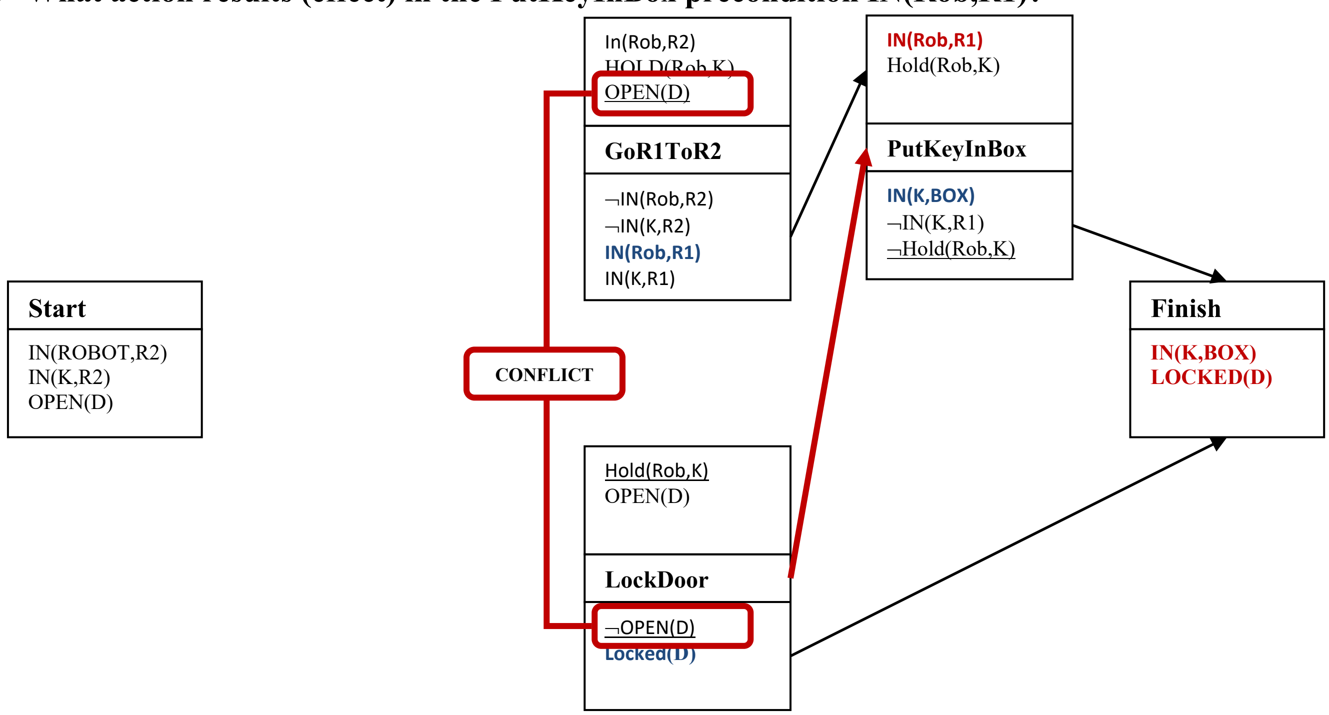


4. **Resolve conflict by placing LockDoor action before PutKeyInBox.**

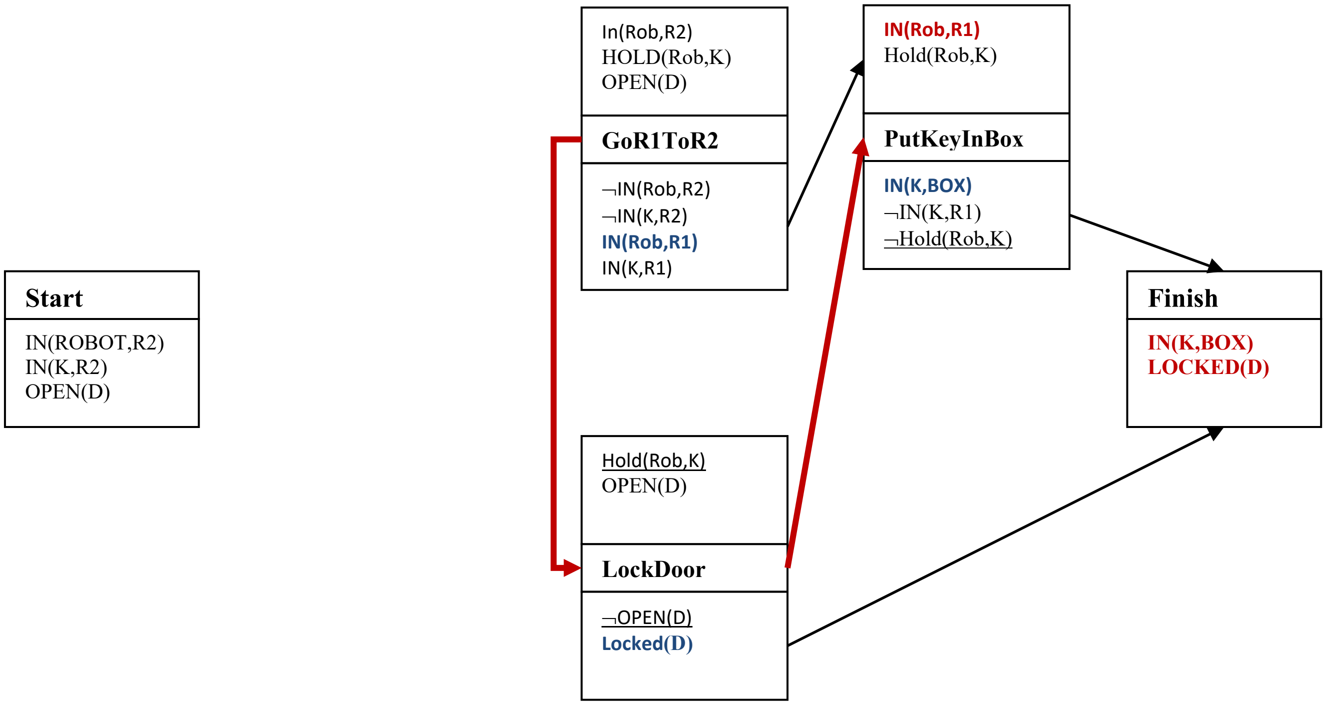


---

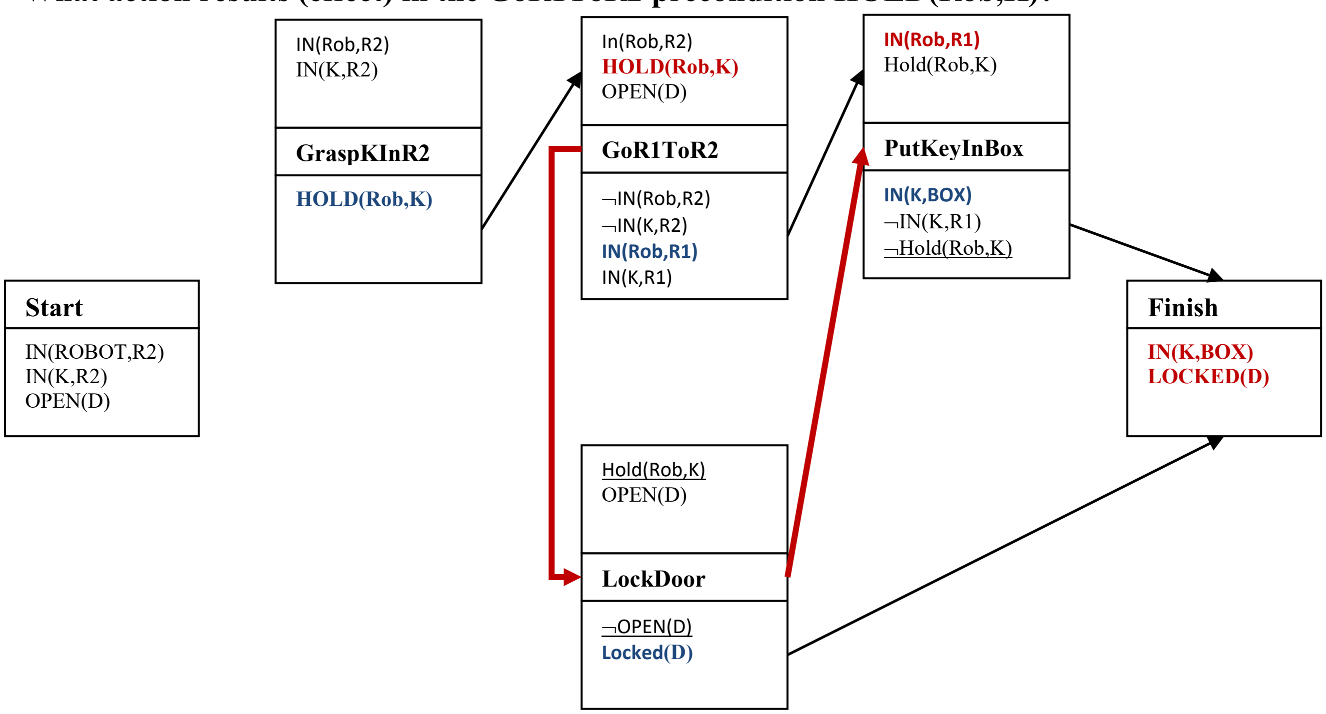
5. **What action results (effect) in the PutKeyInBox precondition IN(Rob,R1)?**



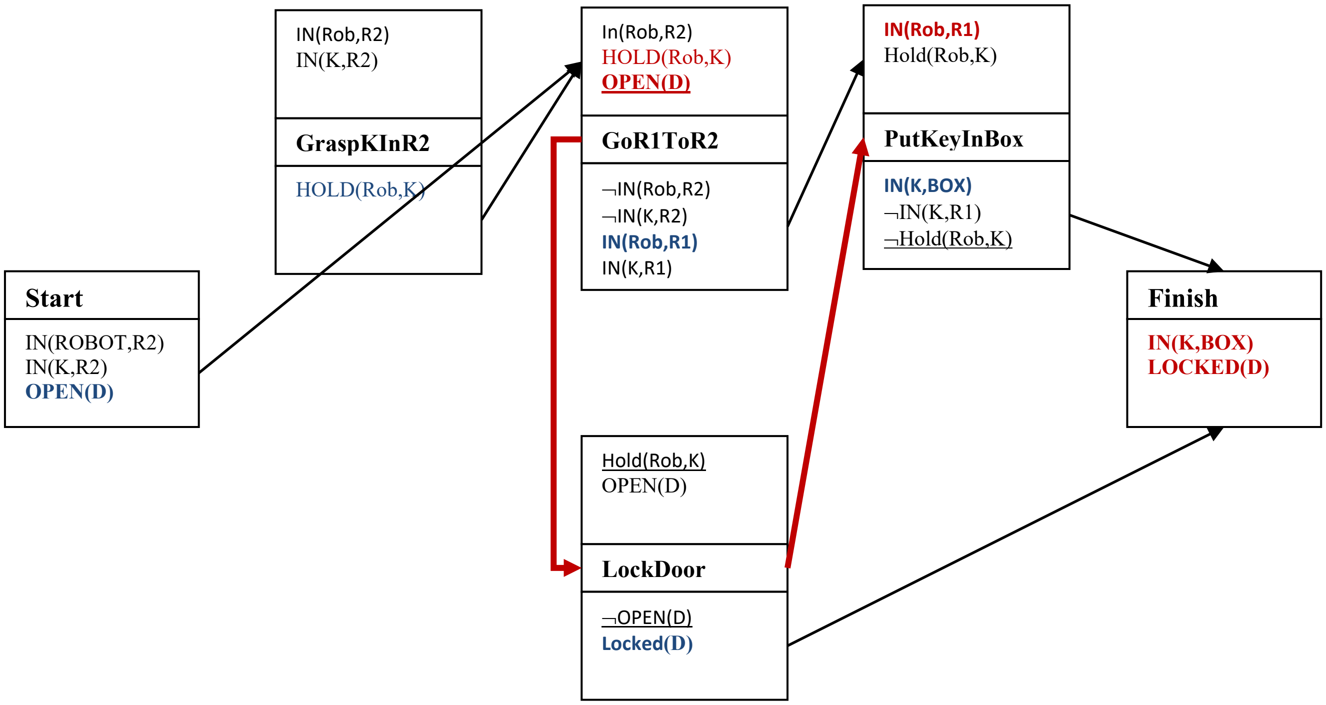
6. Resolve conflict by placing GoR1ToR2 before LockDoor.



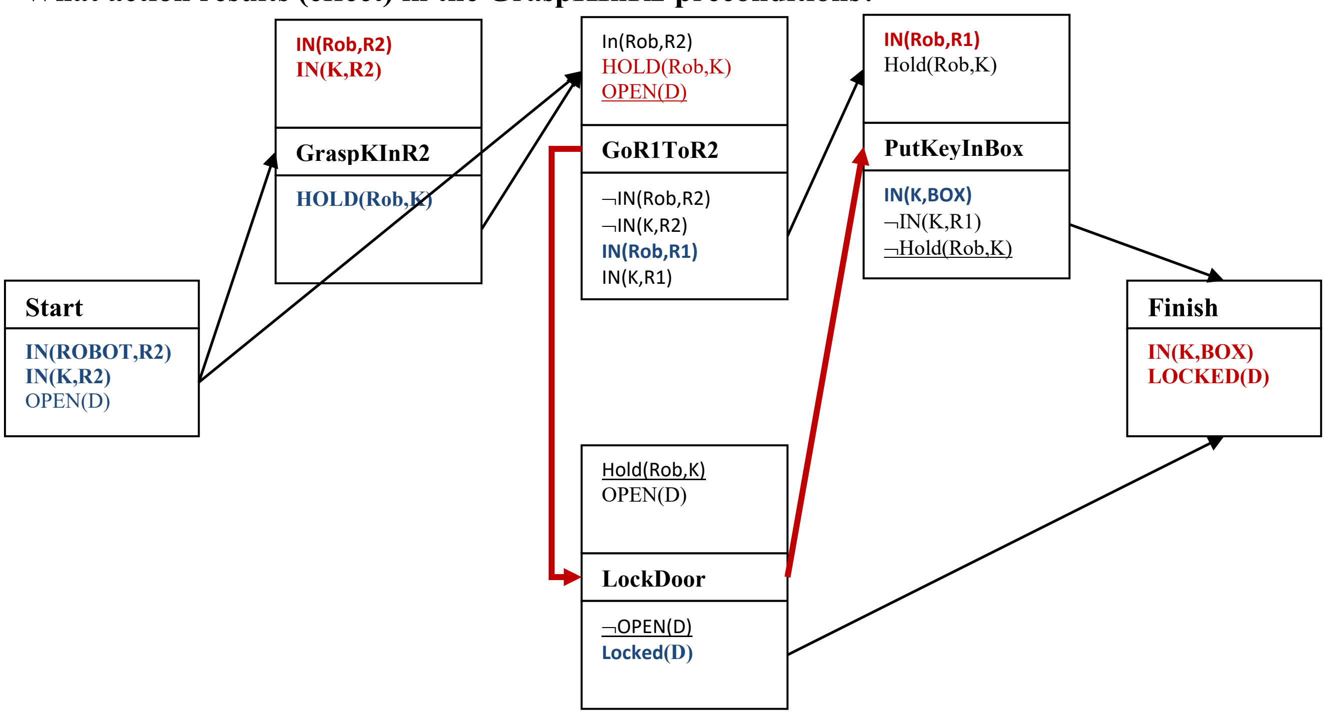
7. What action results (effect) in the GoR1ToR2 precondition HOLD(Rob,K)?



8. What action results (effect) in the GoR1ToR2 precondition OPEN(D)?



9. What action results (effect) in the GraspKInR2 preconditions?



FINAL PLAN:

Start; GraspKInR2; GoR1ToR2; LockDoor; PutKeyInBox; Finish