Fundamentals of Artificial Intelligence

Laboratory

Dr. Mauro Dragoni

Department of Information Engineering and Computer Science
Academic Year 2020/2021

Exercise 10.1

- 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. The result of a ***Grasp*** is that the monkey holds the object if the monkey and object are in the same place at the same height.

  a. Write down the initial state description.

  b. Write the six action schemas.

  c. 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. Write this as a general goal (i.e., not assuming that the box is necessarily at C) in the language of situation calculus. Can this goal be solved by a classical planning system?

Exercise 10.1

a. Write down the initial state description.

$At(Monkey, A) \land$
$At(Bananas, B) \land$
$At(Box, C) \land$
$Height(Monkey, Low) \land$
$Height(Box, Low) \land$
$Height(Bananas, High) \land$
$Pushable(Box) \land$
$Climbable(Box)$

Exercise 10.1

b. Write the six action schemas.

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

Action(ACTION: Push(b, x, y), PRECOND: At(Monkey, x) ∧ Pushable(b),
EFFECT: At(b, y) ∧ At(Monkey, y) ∧ ¬At(b, x) ∧ ¬At(Monkey, x))

Action(ACTION: ClimbUp(b), PRECOND: At(Monkey, x) ∧ At(b, x) ∧ Climbable(b),
EFFECT: On(Monkey, b) ∧ ¬Height(Monkey, High))

Action(ACTION: Grasp(b), PRECOND: Height(Monkey, h) ∧ Height(b, h) ∧ At(Monkey, x) ∧ At(b, x),
EFFECT: Have(Monkey, b))

Action(ACTION: ClimbDown(b), PRECOND: On(Monkey, b) ∧ Height(Monkey, High),
EFFECT: ¬On(Monkey, b) ∧ ¬Height(Monkey, High) ∧ Height(Monkey, Low)

Action(ACTION: UnGrasp(b), PRECOND: Have(Monkey, b),
EFFECT: ¬Have(Monkey, b))

Exercise 10.1

c. 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. Write this as a general goal (i.e., not assuming that the box is necessarily at C) in the language of situation calculus. Can this goal be solved by a classical planning system?

$Have(Monkey, Bananas, s) \wedge (\exists x\ At(Box, x, s0) \wedge At(Box, x, s))$

In STRIPS, we can only talk about the goal state; there is no way of representing the fact that there must be some relation (such as equality of location of an object) between two states within the plan. So there is no way to represent this goal.

Exercise 10.2

- The figure shows a version of Shakey's world consisting of four rooms lined up along a corridor, where each room has a door and a light switch. The actions in Shakey's world include moving from place to place, pushing movable objects (such as boxes), climbing onto and down from rigid objects (such as boxes), and turning light switches on and off. The robot itself could not climb on a box or toggle a switch, but the planner was capable of finding and printing out plans that were beyond the robot's abilities. Shakey's six actions are the following:

  - *Go(x, y, r)*, which requires that Shakey be **At x** and that **x** and **y** are locations **In** the same room **r**. By convention a door between two rooms is in both of them.

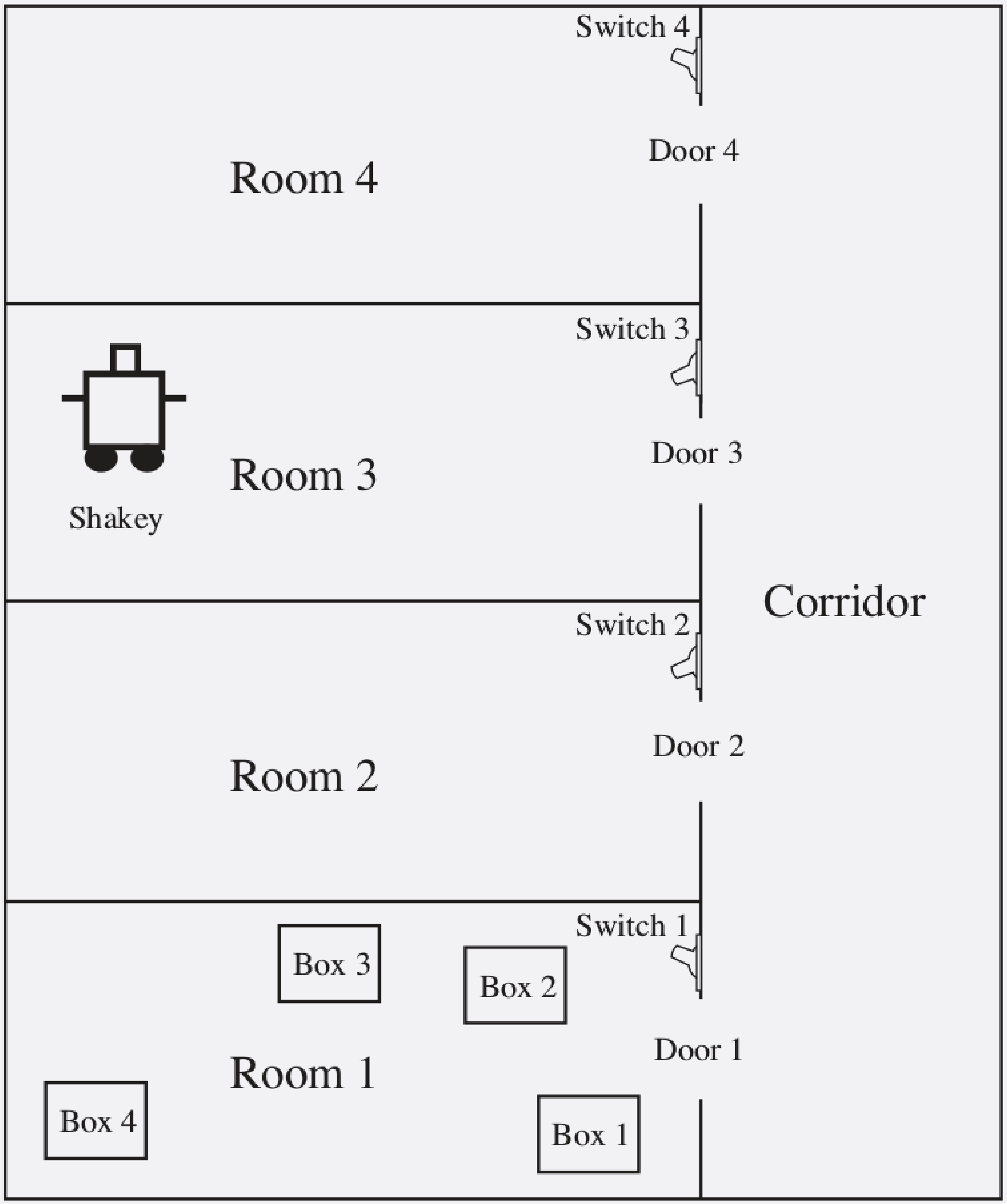
  - Push a box **b** from location **x** to location **y** within the same room: *Push(b, x, y, r)*. You will need the predicate **Box** and constants for the boxes.

  - Climb onto a box from position **x**: *ClimbUp(x, b)*; climb down from a **box** to position **x**: *ClimbDown(b, x)*. We will need the predicate **On** and the constant **Floor**.

  - Turn a light switch on or off: *TurnOn(s, b)*; *TurnOff(s, b)*. To turn a light on or off, Shakey must be on top of a box at the light switch's location.

Write PDDL sentences for Shakey's six actions and the initial state from the Figure. Construct a plan for Shakey to get Box2 into Room2 .

Exercise 10.2



Exercise 10.2

Initial state:

$In(Switch1, Room1) \land In(Door1, Room1) \land In(Door1, Corridor)$
$In(Switch1, Room2) \land In(Door2, Room2) \land In(Door2, Corridor)$
$In(Switch1, Room3) \land In(Door3, Room3) \land In(Door3, Corridor)$
$In(Switch1, Room4) \land In(Door4, Room4) \land In(Door4, Corridor)$
$In(Shakey, Room3) \land At(Shakey, X_S)$
$In(Box1, Room1) \land In(Box2, Room1) \land In(Box3, Room1) \land In(Box4, Room1)$
$Climbable(Box1) \land Climbable(Box2) \land Climbable(Box3) \land Climbable(Box4)$
$Pushable(Box1) \land Pushable(Box2) \land Pushable(Box3) \land Pushable(Box4)$
$At(Box1, X_1) \land At(Box2, X_2) \land At(Box3, X_3) \land At(Box4, X_4)$
$TurnedOn(Switch1) \land TurnedOn(Switch4)$

Exercise 10.2

Actions:

$Action(ACTION: Go(x, y), PRECOND: At(Shakey, x) \wedge In(x, r) \wedge In(y, r),$
$\quad EFFECT: At(Shakey, y) \wedge \neg(At(Shakey, x)))$

$Action(ACTION: Push(b, x, y), PRECOND: At(Shakey, x) \wedge Pushable(b),$
$\quad EFFECT: At(b, y) \wedge At(Shakey, y) \wedge \neg At(b, x) \wedge \neg At(Shakey, x))$

$Action(ACTION: ClimbUp(b), PRECOND: At(Shakey, x) \wedge At(b, x) \wedge Climbable(b),$
$\quad EFFECT: On(Shakey, b) \wedge \neg On(Shakey, Floor))$

$Action(ACTION: ClimbDown(b), PRECOND: On(Shakey, b),$
$\quad EFFECT: On(Shakey, Floor) \wedge \neg On(Shakey, b))$

$Action(ACTION: TurnOn(l), PRECOND: On(Shakey, b) \wedge At(Shakey, x) \wedge At(l, x),$
$\quad EFFECT: TurnedOn(l))$

$Action(ACTION: TurnOff(l), PRECOND: On(Shakey, b) \wedge At(Shakey, x) \wedge At(l, x),$
$\quad EFFECT: \neg TurnedOn(l))$

Exercise 10.2

Plan:

Go(X_S, *Door3*)
Go(*Door3*, *Door1*)
Go(*Door1*, X_2)
Push(*Box2*, X_2, *Door1*)
Push(*Box2*, *Door1*, *Door2*)
Push(*Box2*, *Door2*, *Switch2*)

Exercise 10.3

- A finite Turing machine has a finite one-dimensional tape of cells, each cell containing one of a finite number of symbols. One cell has a read and write head above it. There is a finite set of states the machine can be in, one of which is the accept state. At each time step, depending on the symbol on the cell under the head and the machine's current state, there are a set of actions we can choose from. Each action involves writing a symbol to the cell under the head, transitioning the machine to a state, and optionally moving the head left or right. The mapping that determines which actions are allowed is the Turing machine's program. Your goal is to control the machine into the accept state. Represent the Turing machine acceptance problem as a planning problem. If you can do this, it demonstrates that determining whether a planning problem has a solution is at least as hard as the Turing acceptance problem, which is PSPACE-hard.

Exercise 10.3

One possible representation.

- *HeadAt(c)*: tape head at cell location c, true for exactly one cell.
- *State(s)*: machine state is s, true for exactly one cell.
- *ValueOf(c, v)*: cell c's value is v.
- *LeftOf(c1, c2)*: cell c1 is one step left from cell c2 .
- *TransitionLeft(s1, v1, s2, v2)*: the machine in state s1 upon reading a cell with value v1 may write value v2 to the cell, change state to s2 , and transition to the left.
- *TransitionRight(s1, v1, s2, v2)*: the machine in state s1 upon reading a cell with value v1 may write value v2 to the cell, change state to s2, and transition to the right.

The predicates *HeadAt*, *State*, and *ValueOf* are fluents, the rest are constant descriptions of the machine and its tape.

Exercise 10.3

Actions:

Action(RunLeft(s1, c1, v1, s2, c2, v2),
PRECOND: State(s1) ∧ HeadAt(c1) ∧ ValueOf(c1, v1) ∧ TransitionLeft(s1, v1, s2, v2) ∧ LeftOf(c2, c1)
EFFECT: ¬State(s1) ∧ State(s2) ∧ ¬HeadAt(c1) ∧ HeadAt(c2) ∧ ¬ValueOf(c1, v1) ∧ ValueOf(c1, v2))

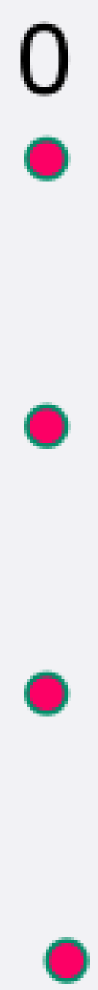
Action(RunRight(s1, c1, v1, s2, c2, v2),
PRECOND: State(s1) ∧ HeadAt(c1) ∧ ValueOf(c1, v1) ∧ TransitionRight(s1, v1, s2, v2) ∧ LeftOf(c1, c2)
EFFECT: ¬State(s1) ∧ State(s2) ∧ ¬HeadAt(c1) ∧ HeadAt(c2) ∧ ¬ValueOf(c1, v1) ∧ ValueOf(c1, v2))

The goal will typically be to reach a fixed accept state. A simple example problem is:

Init(HeadAt(C0) ∧ State(S1) ∧ ValueOf(C0, 1) ∧ ValueOf(C1, 1) ∧ ValueOf(C2, 1) ∧ ValueOf(C3, 0) ∧ LeftOf(C0, C1) ∧ LeftOf(C1, C2) ∧ LeftOf(C2, C3) ∧ TransitionLeft(S1, 1, S1, 0) ∧ TransitionLeft(S1, 0, Saccept, 0)

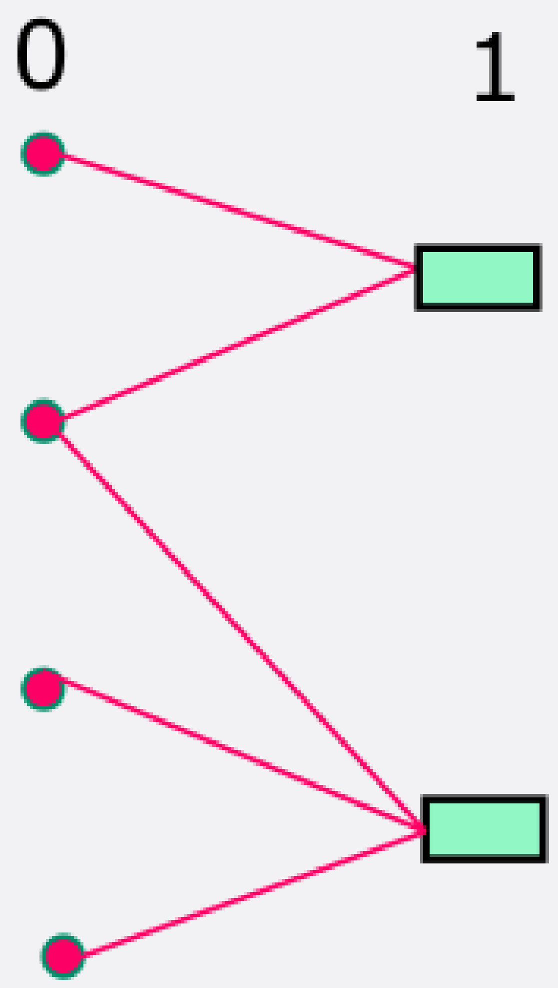
Goal(State(Saccept))

Recap planning graphs



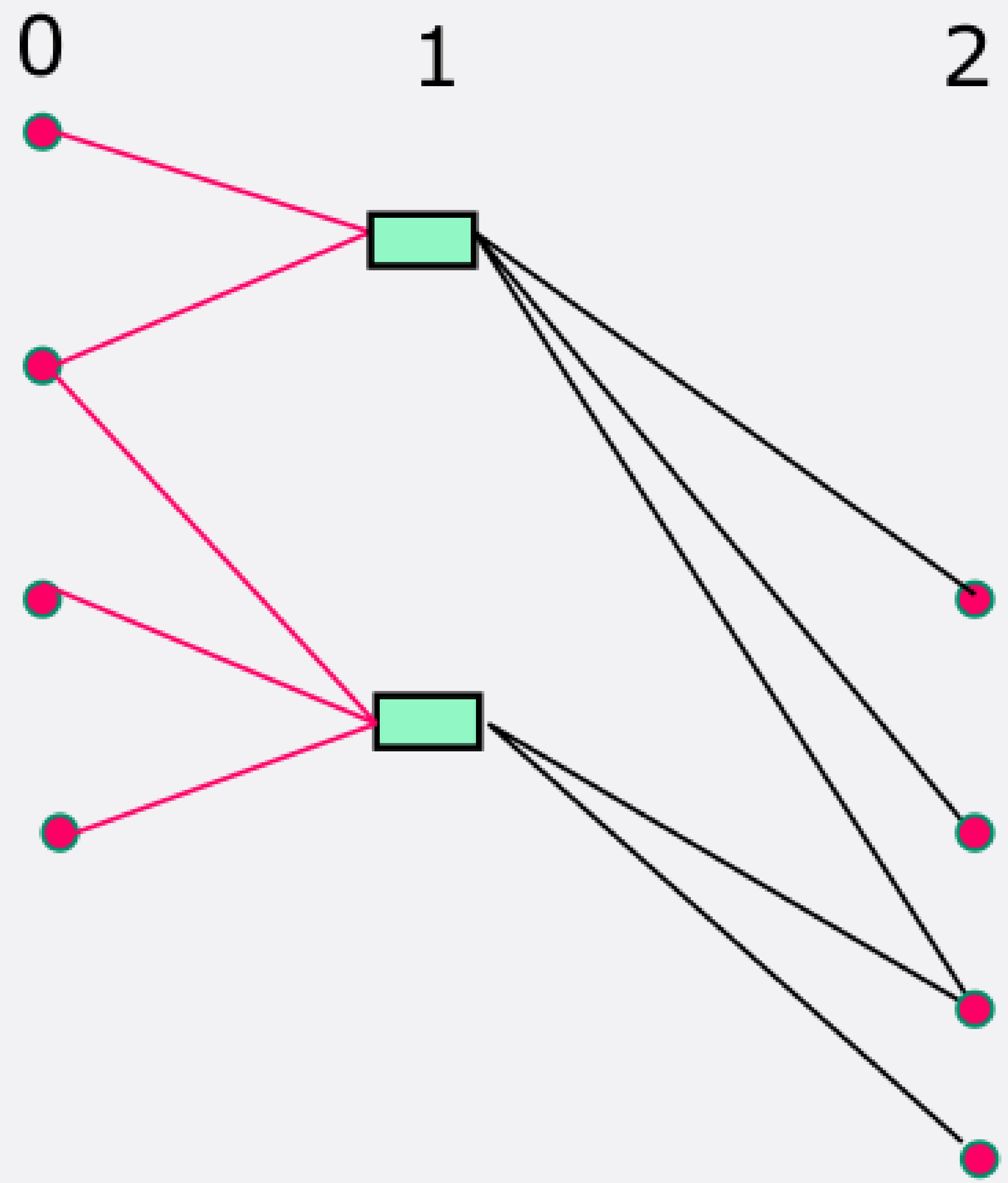
- Start with initial conditions

Recap planning graphs



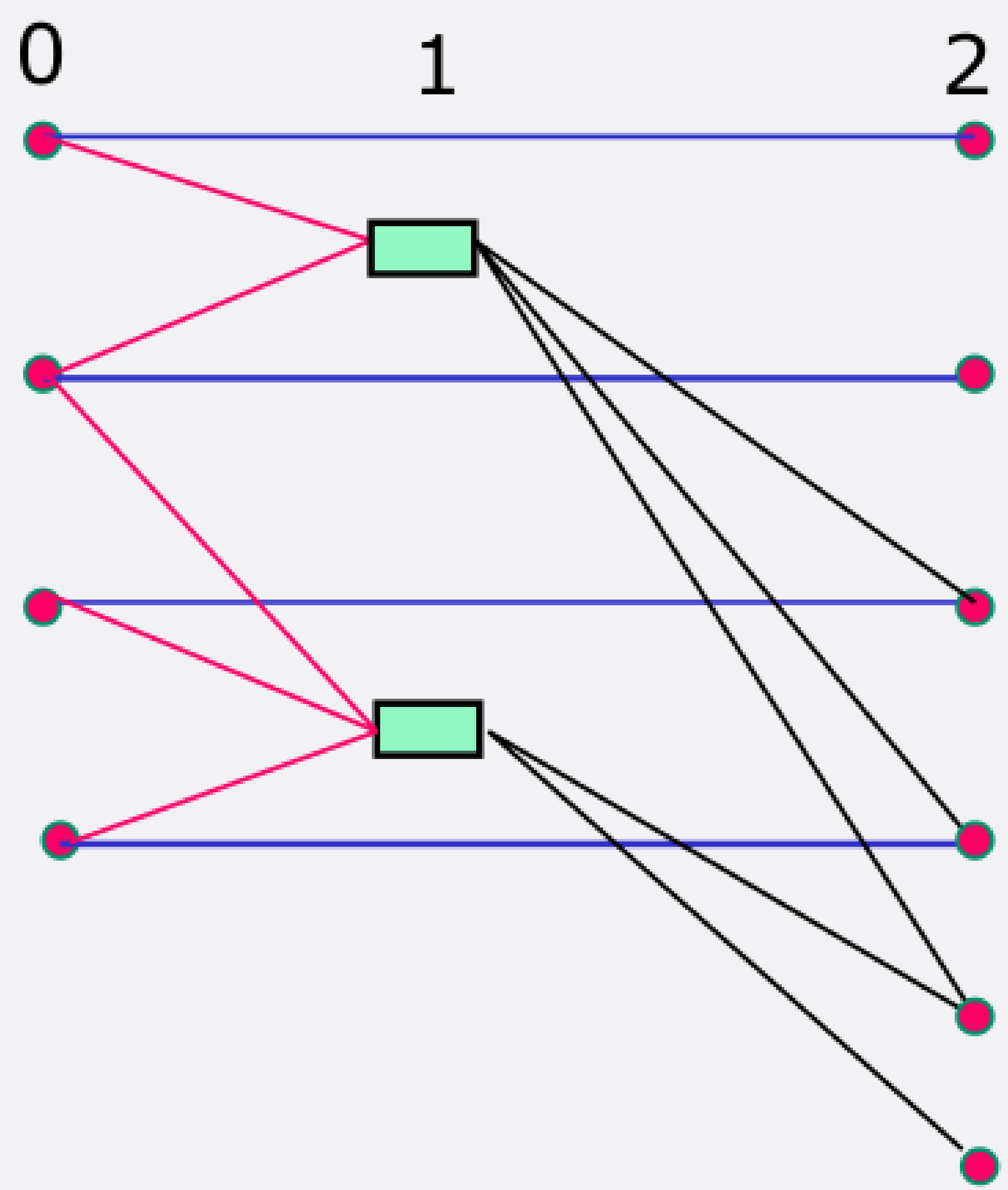
- Start with initial conditions
- Add actions with satisfied preconditions

Recap planning graphs



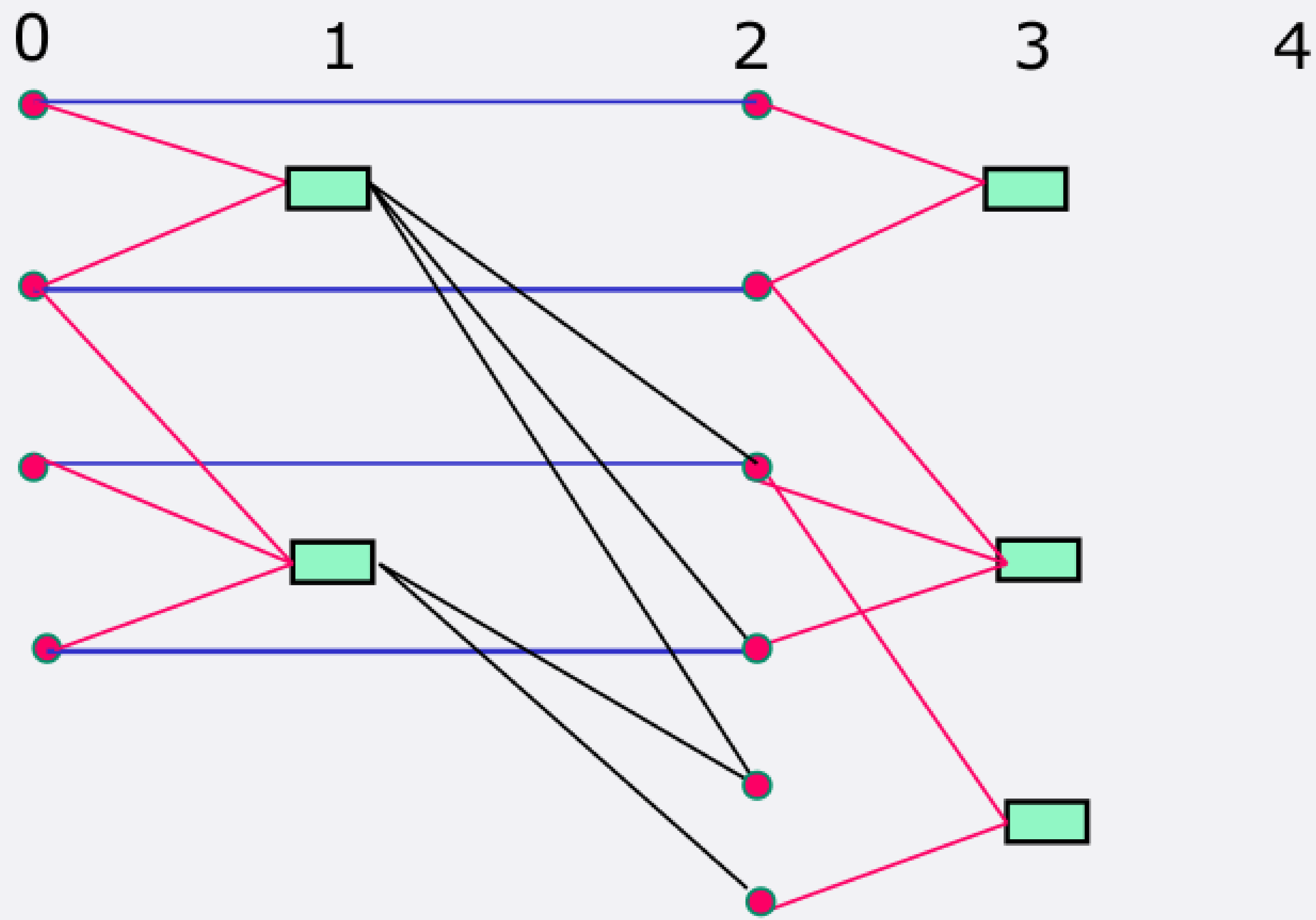
- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels

Recap planning graphs



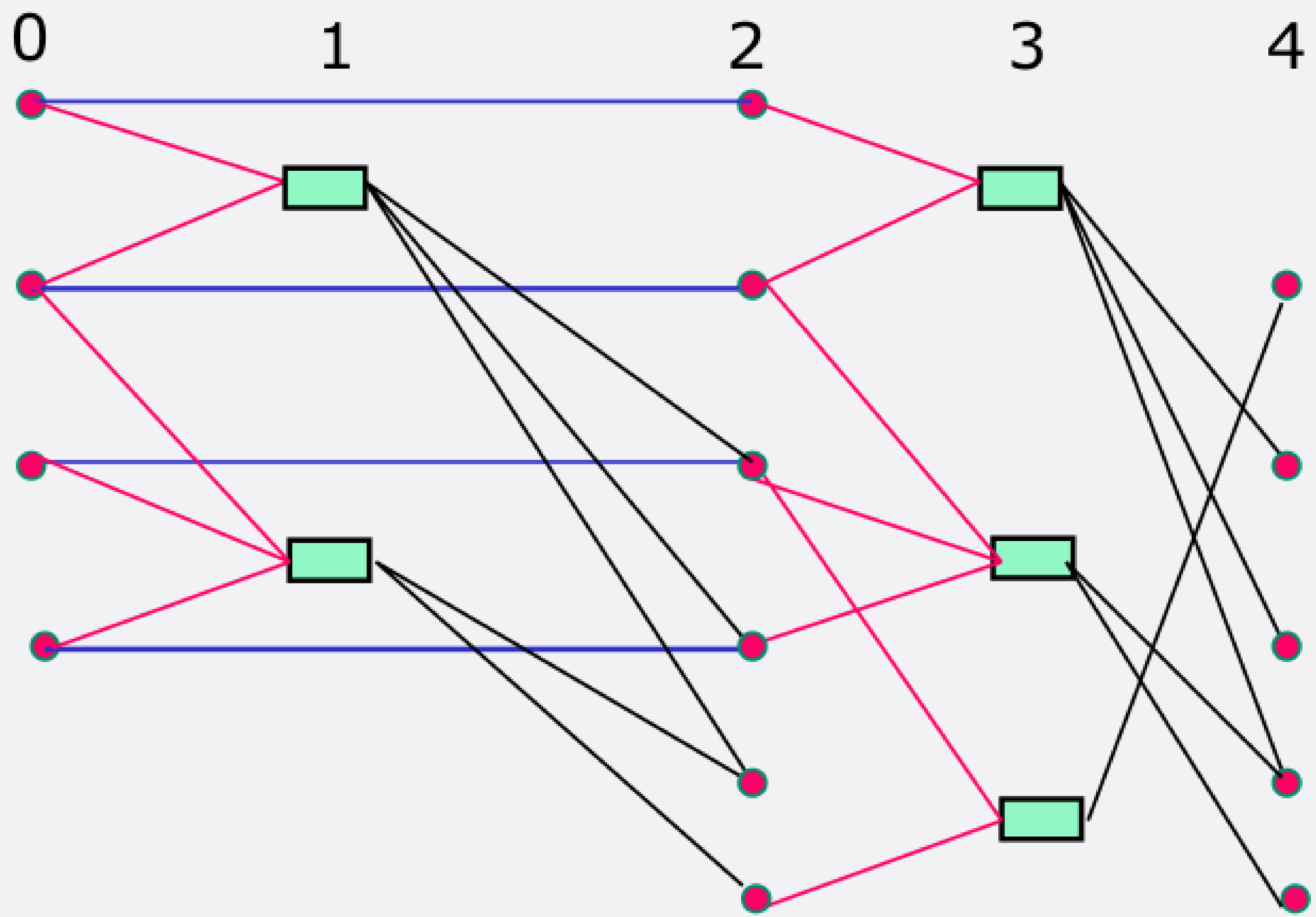
- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels
- Add maintenance actions: these maintenance actions represent the possibility of having some proposition be true at step n because it was true at step n-2, and we didn't do anything to make it false; that is, that we maintained its truth value.

Recap planning graphs



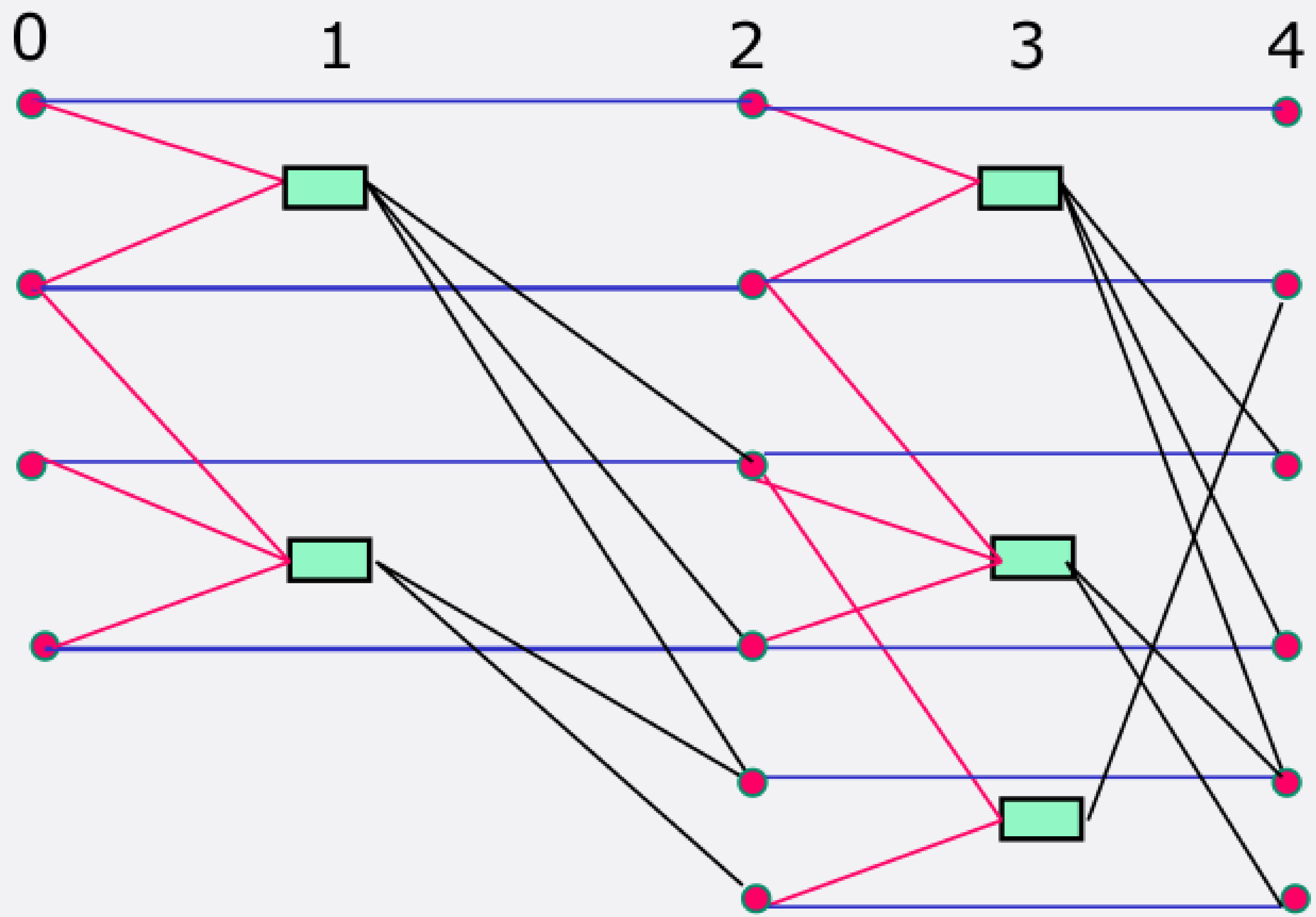
- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels
- Add maintenance actions: these maintenance actions represent the possibility of having some proposition be true at step n because it was true at step n-2, and we didn't do anything to make it false; that is, that we maintained its truth value.

Recap planning graphs



- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels
- Add maintenance actions: these maintenance actions represent the possibility of having some proposition be true at step n because it was true at step n-2, and we didn't do anything to make it false; that is, that we maintained its truth value.

Recap planning graphs



- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels
- Add maintenance actions: these maintenance actions represent the possibility of having some proposition be true at step n because it was true at step n-2, and we didn't do anything to make it false; that is, that we maintained its truth value.

Recap planning graphs

- A **mutex relation** holds between **two actions** when:
  - Inconsistent effects: one action negates the effect of another.
  - Interference: one of the effects of one action is the negation of a precondition of the other.
  - Competing needs: one of the preconditions of one action is mutually exclusive with the precondition of the other.

- A **mutex relation** holds between **two literals** when:
  - one is the negation of the other, or
  - each possible action pair that could achieve the literals is mutex (inconsistent support).

Exercise 10.4

Cake Example

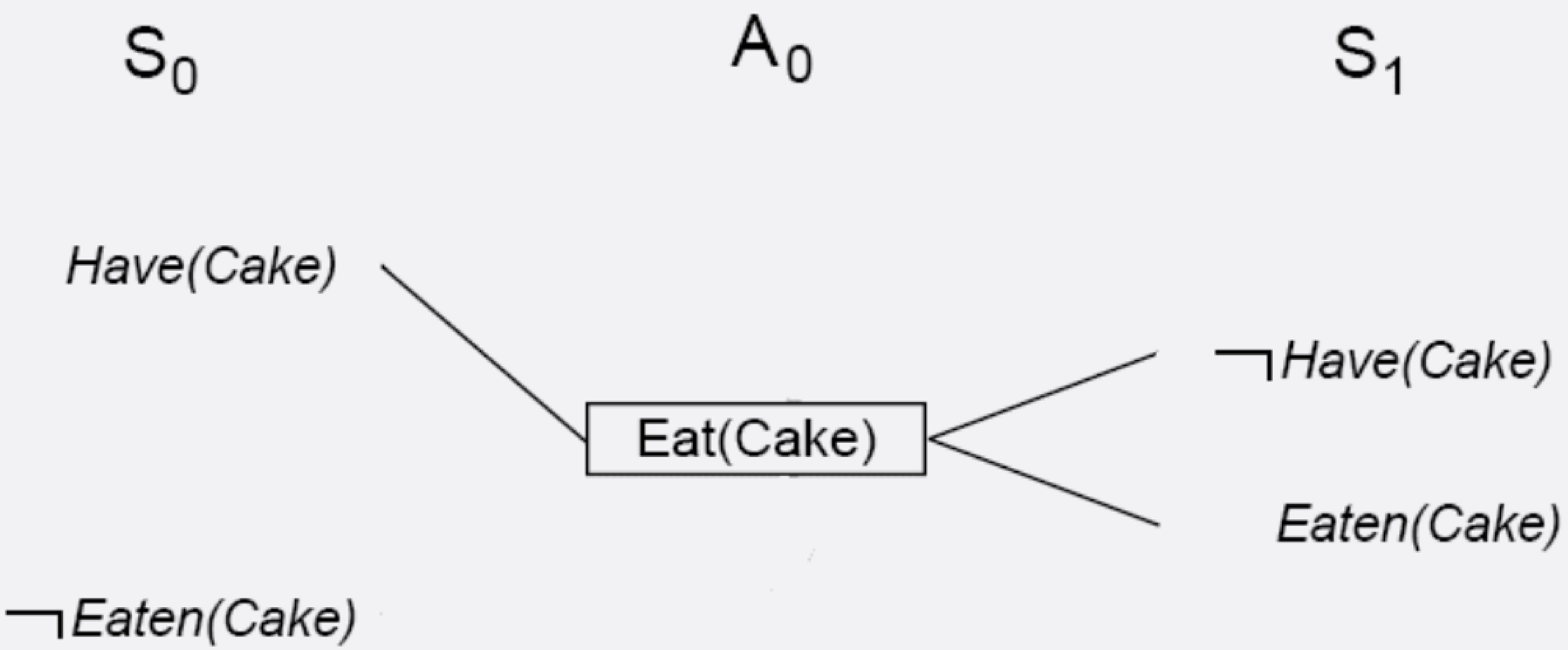
- $Init(Have(Cake))$
- $Goal(Have(Cake) \land Eaten(Cake))$
- $Action(Eat(Cake),$
  - $PRECOND: Have(Cake)$
  - $EFFECT: \neg Have(Cake) \land Eaten(Cake))$
- $Action(Bake(Cake),$
  - $PRECOND: \neg Have(Cake)$
  - $EFFECT: Have(Cake))$

Exercise 10.4

S_0	A_0	S_1
Have(Cake)		
$\neg$*Eaten(Cake)*		

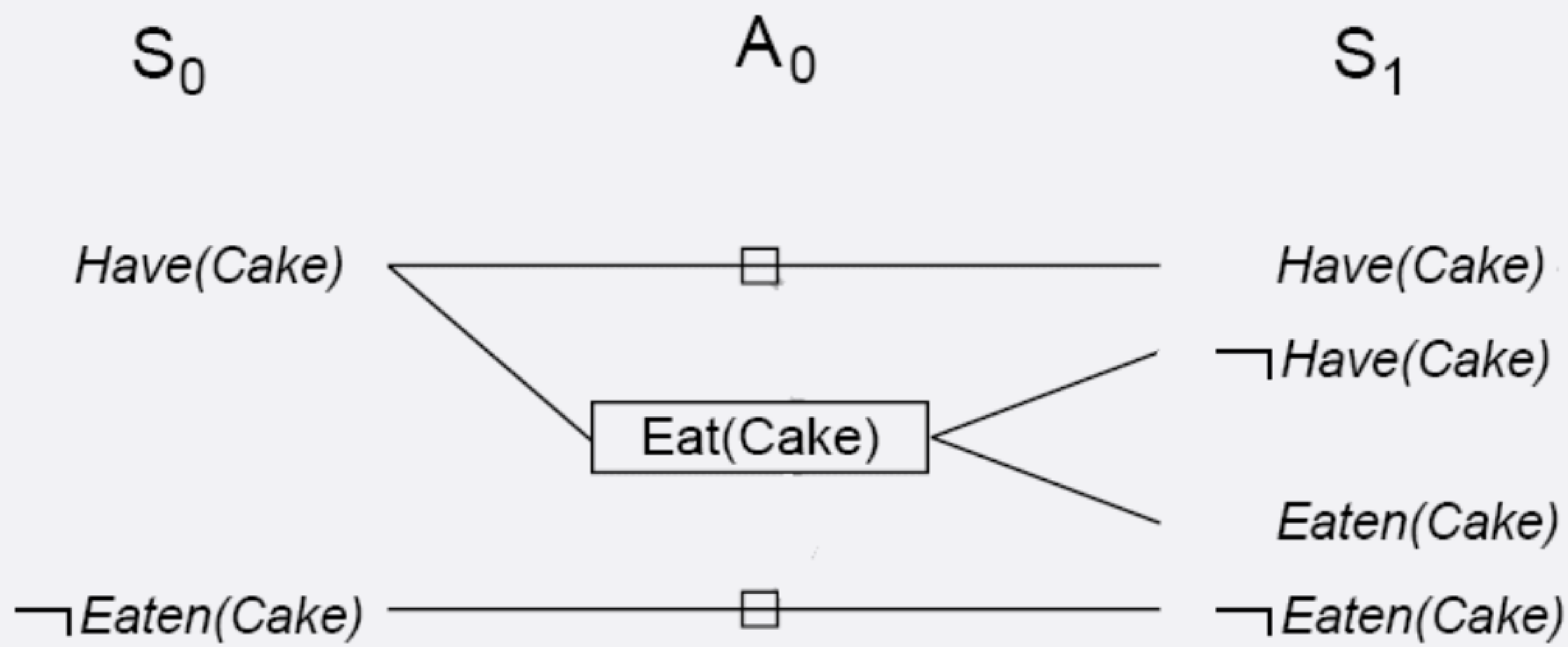
Create level 0 from initial problem state.

Exercise 10.4



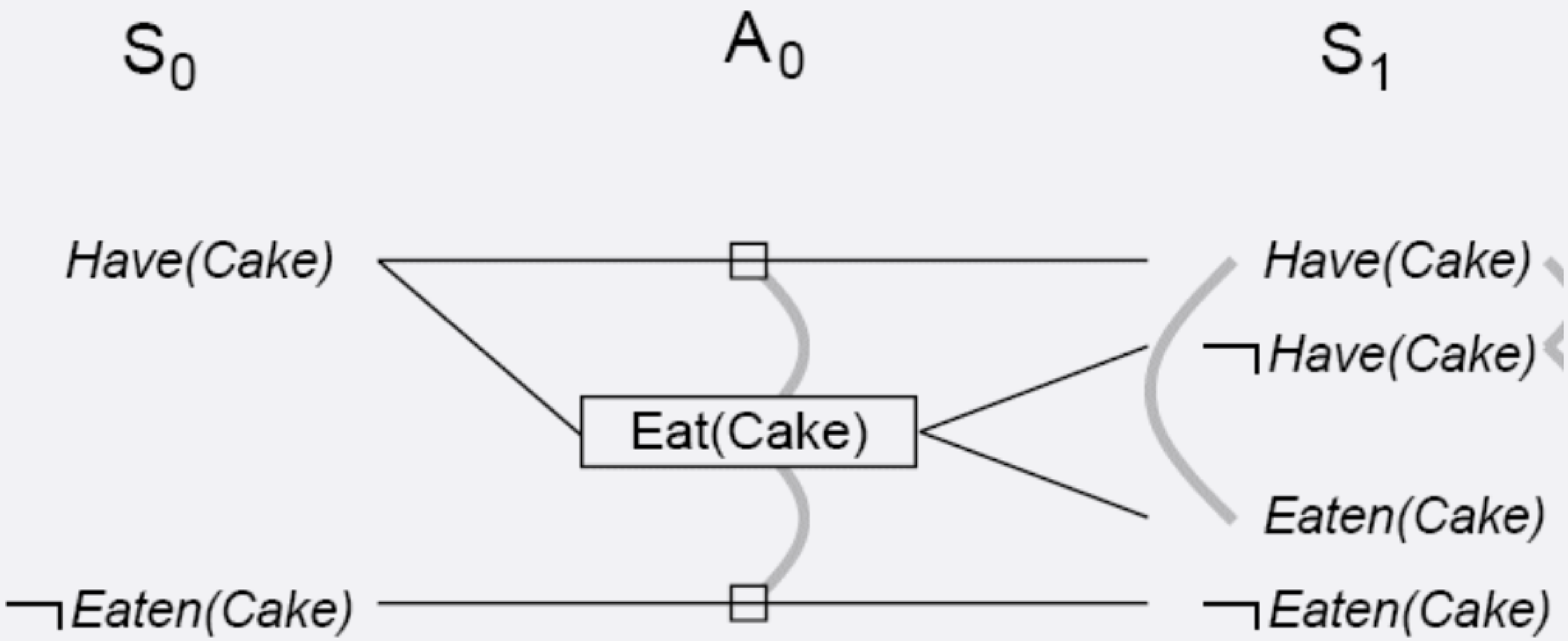
Add all applicable actions.
Add all effects to the next state.

Exercise 10.4



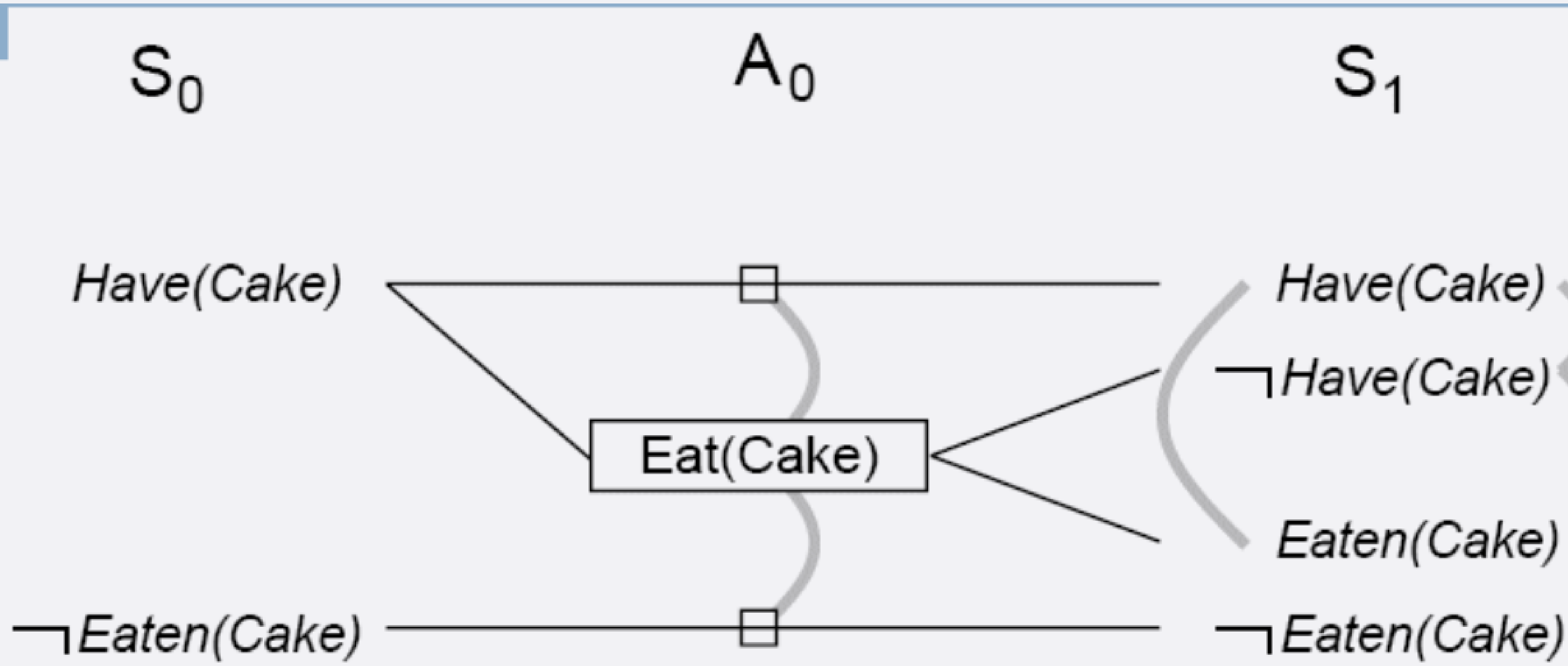
Add persistence actions to map all literals in state S_i to state S_{i+1}

Exercise 10.4



Identify mutual exclusions between actions and literals based on potential conflicts.

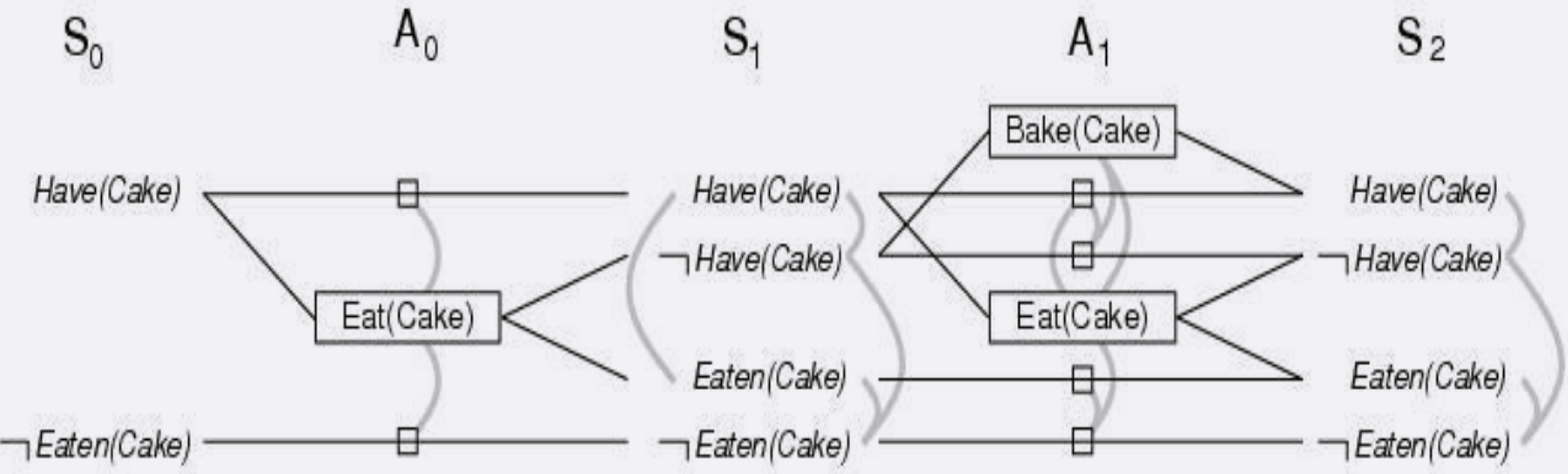
Exercise 10.4



Level S_1 contains all literals that could result from picking any subset of actions in A_0

- Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
- S_1 defines multiple states and the mutex links are the constraints that define this set of states.

Exercise 10.4



Repeat process until graph levels off:

- two consecutive levels are identical, or
- contains the same amount of literals.

Exercise 10.5

Birthday Dinner Example

- *Goal: ¬Garb ∧ Dinner ∧ Present*
- *Init: Garb ∧ Clean ∧ Quiet*

- *Actions:*
  - *Cook - Pre: Clean - Effect: Dinner*
  - *Wrap - Pre: Quiet - Effect: Present*
  - *Carry - Pre: Garb - Effect: ¬Garb ∧ ¬Clean*
  - *Dolly - Pre: Garb - Effect: ¬Garb ∧ ¬Quiet*

Exercise 10.5

clean •

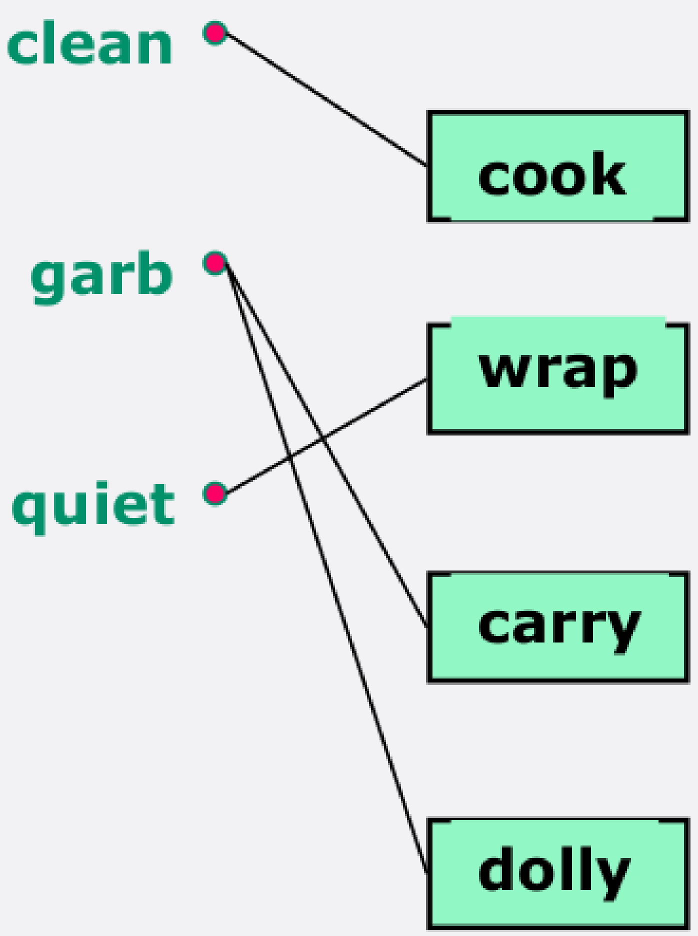
garb •

quiet •

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We start by putting in the initial conditions.

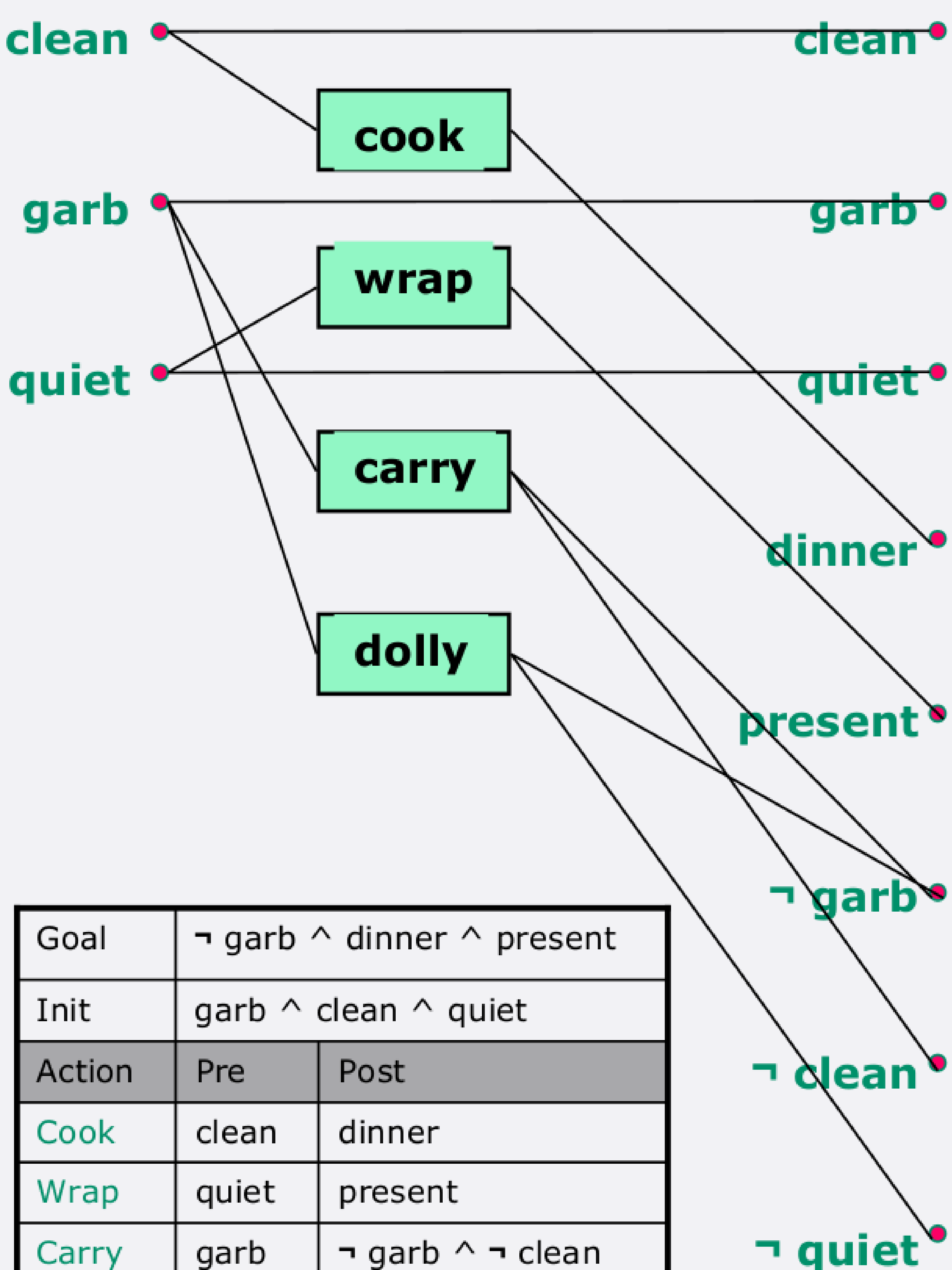
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Given those initial conditions, all four of our actions could possibly be executed on the first step, so we add them to the graph.

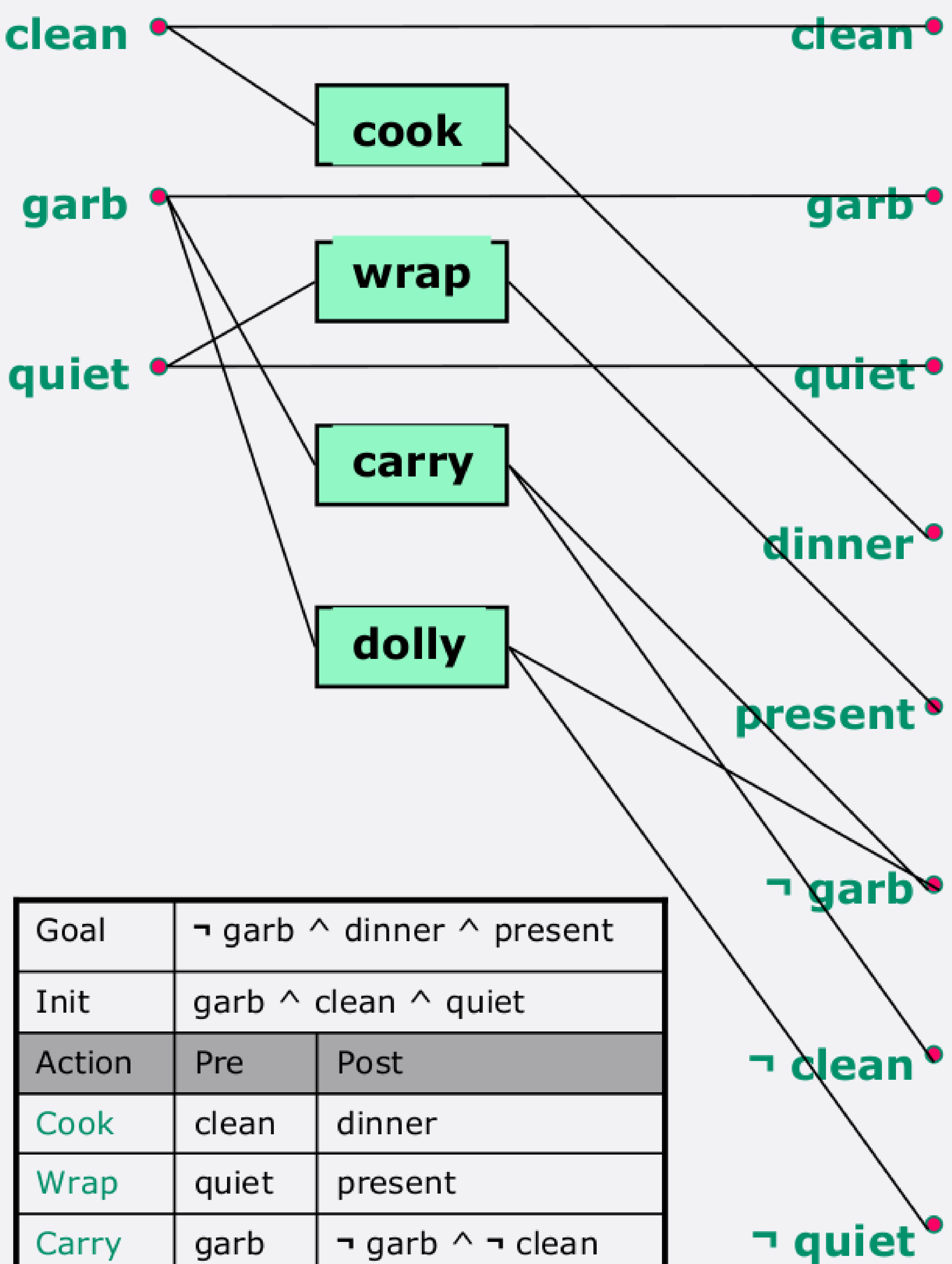
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now we add all of our old propositions to the next layer, as well as all the propositions that could be effects of the actions. And we draw in the maintenance actions, as well.

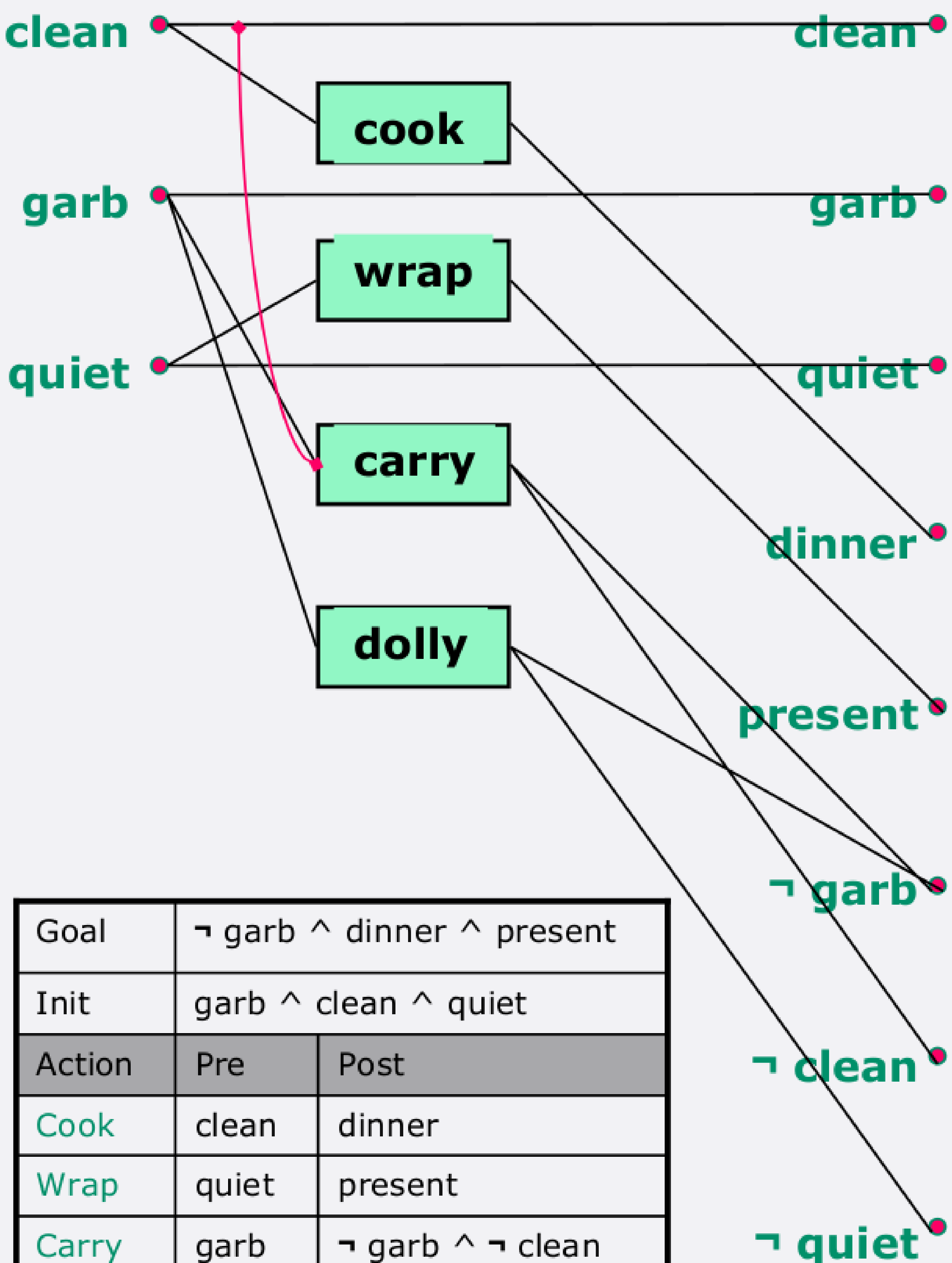
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now it's time to do the mutexes. None of the initial propositions are mutex (or we're starting in an impossible state). So, let's look at the actions in layer 1.

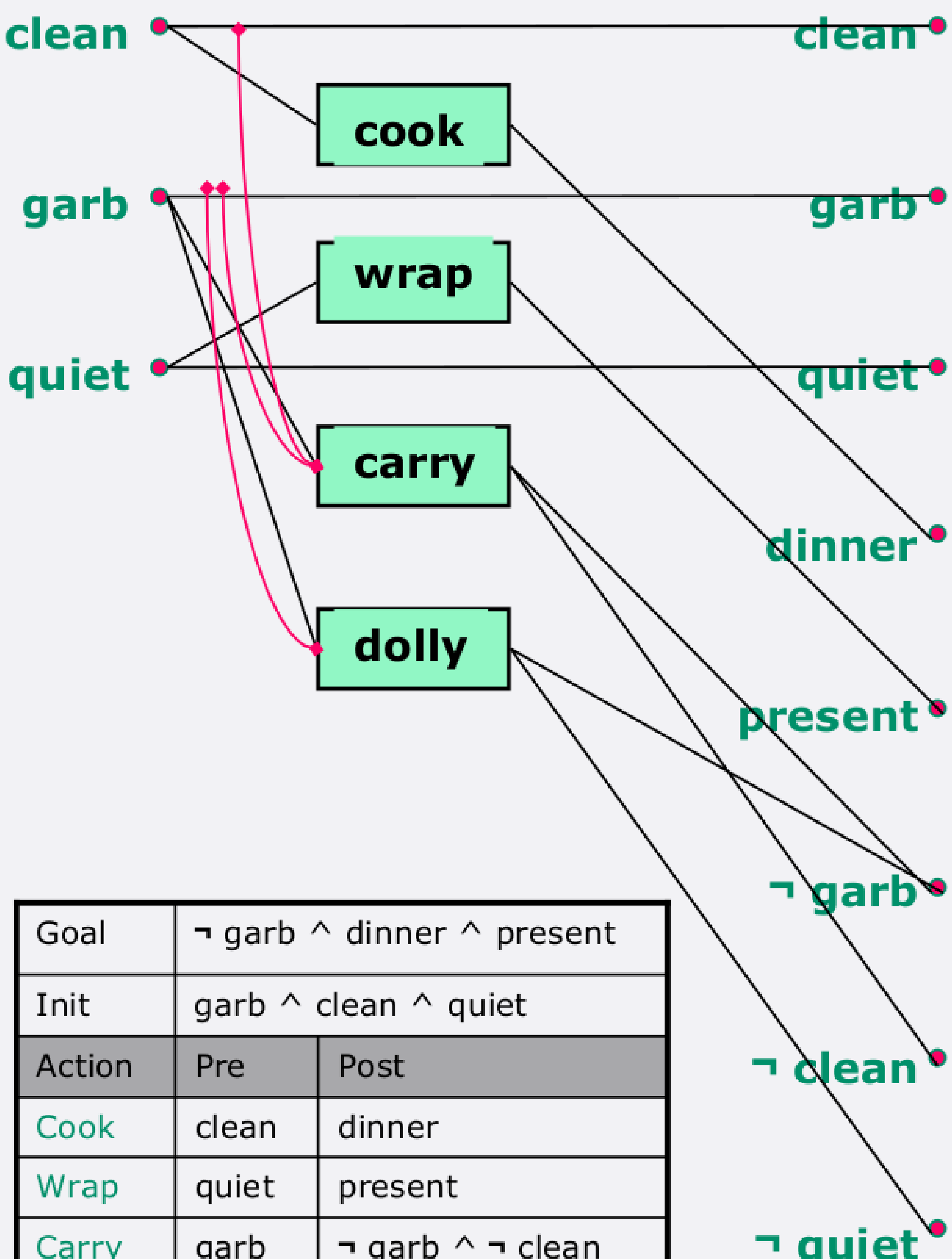
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

The first reason that actions can be mutex is due to inconsistent effects. So, **carry** and **maintaining clean** have inconsistent effects (because carry makes clean false).

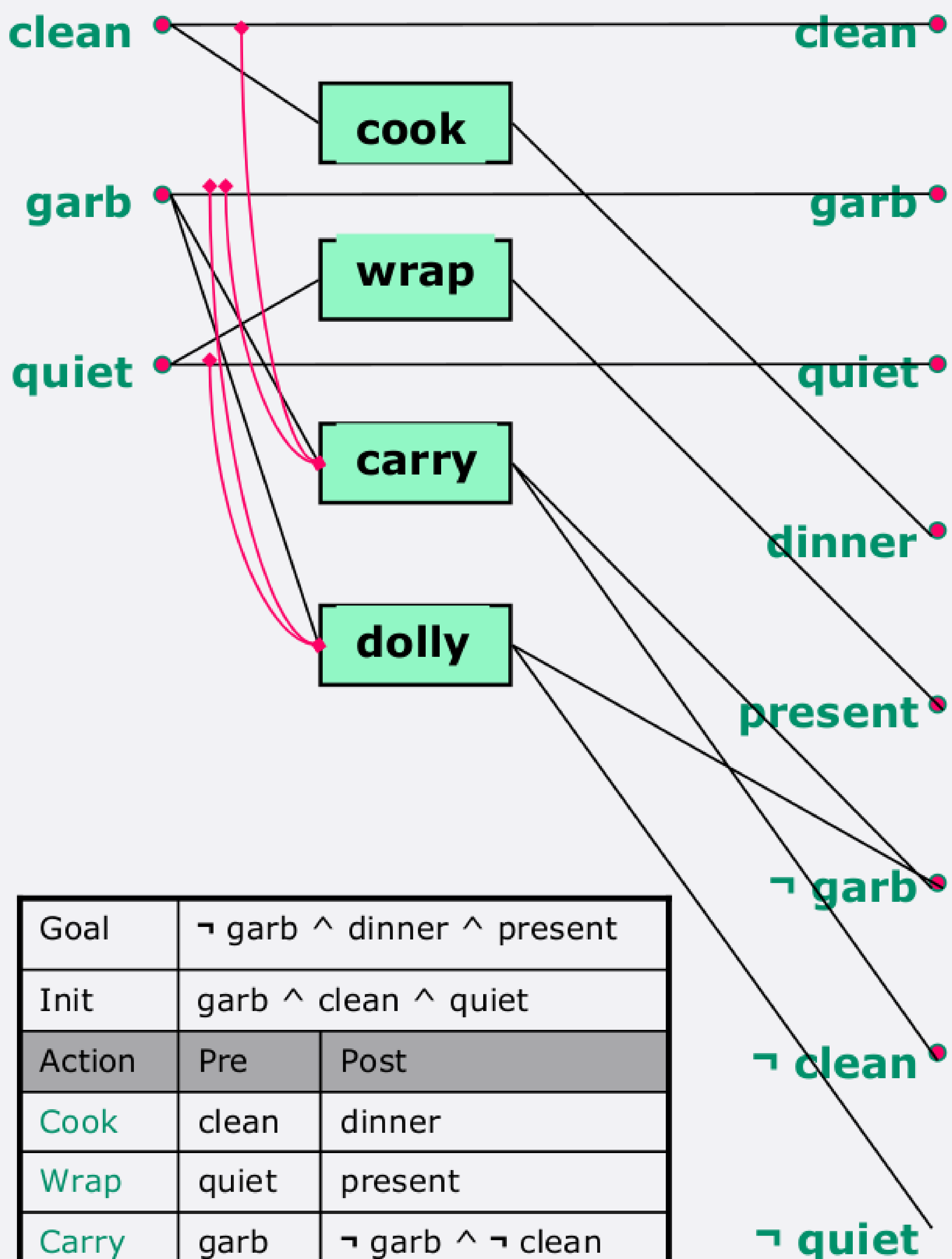
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And **maintaining garb** has inconsistent effects with both **carry** and **dolly** (which make garb false).

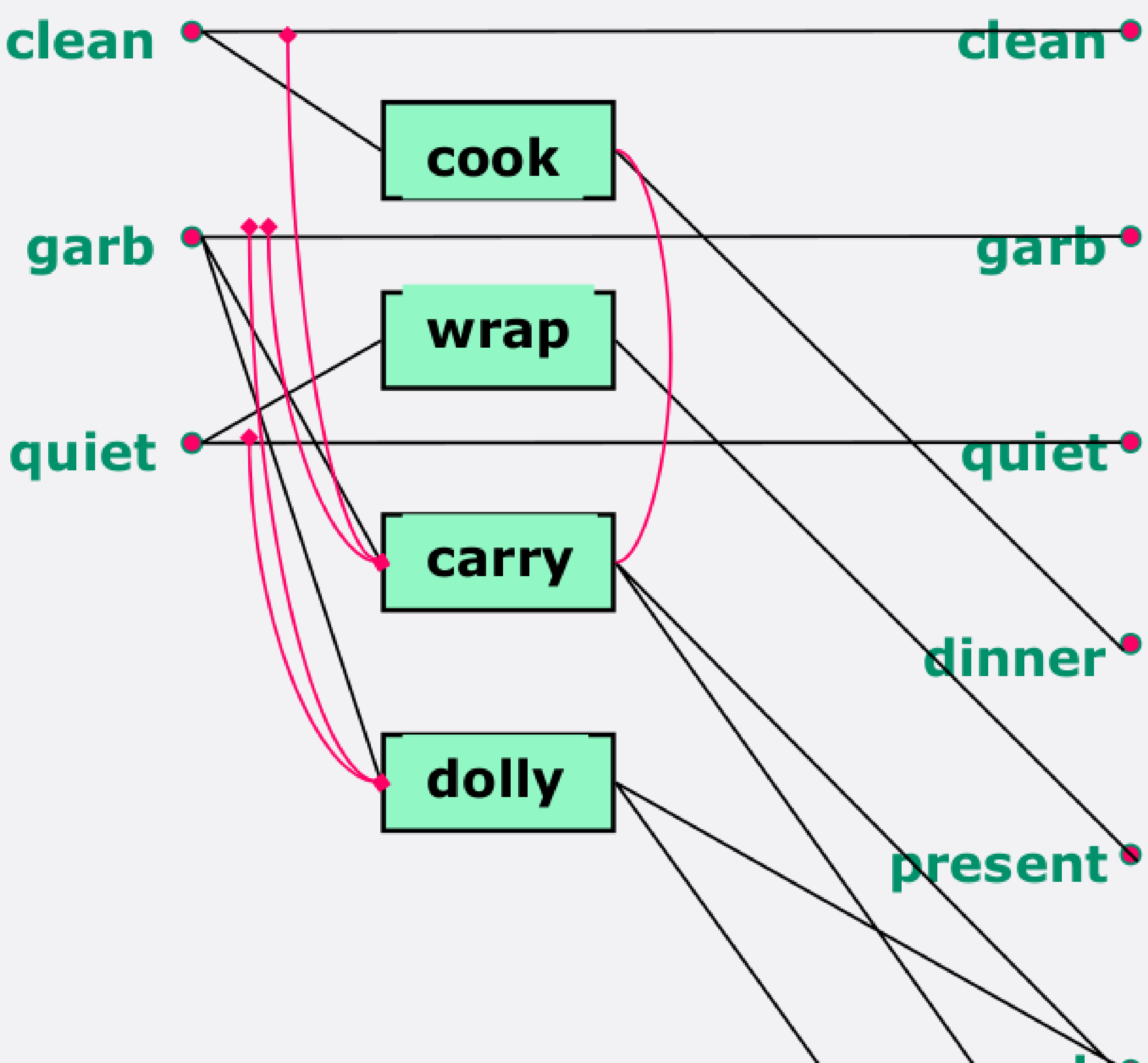
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And **maintaining quiet** has inconsistent effects with **dolly** (which makes quiet false).

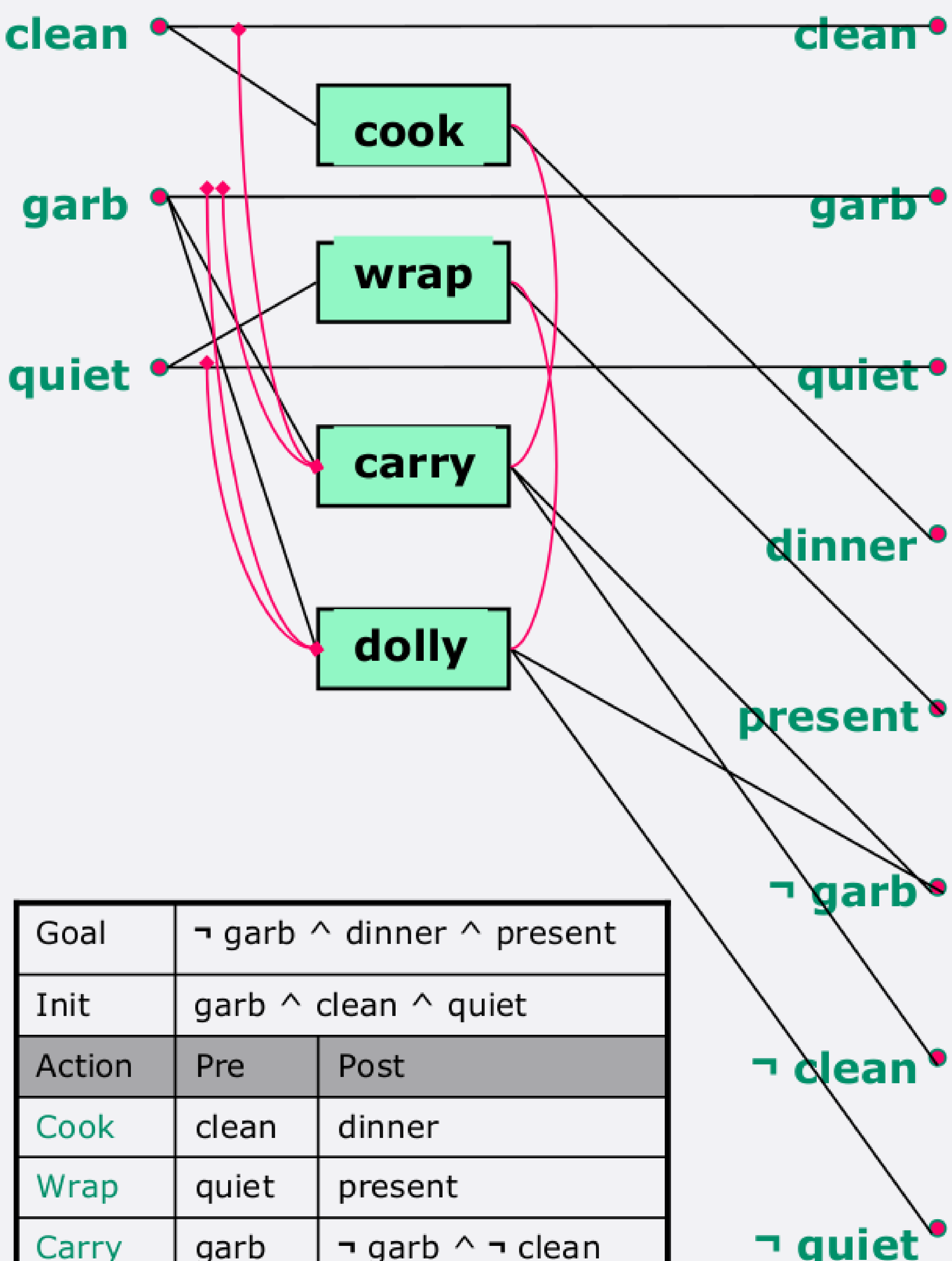
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Another kind of mutex is due to interference: one action negates the precondition of another. Here we have interference between **cook** and **carry** (carry makes clean false, which is required for cook).

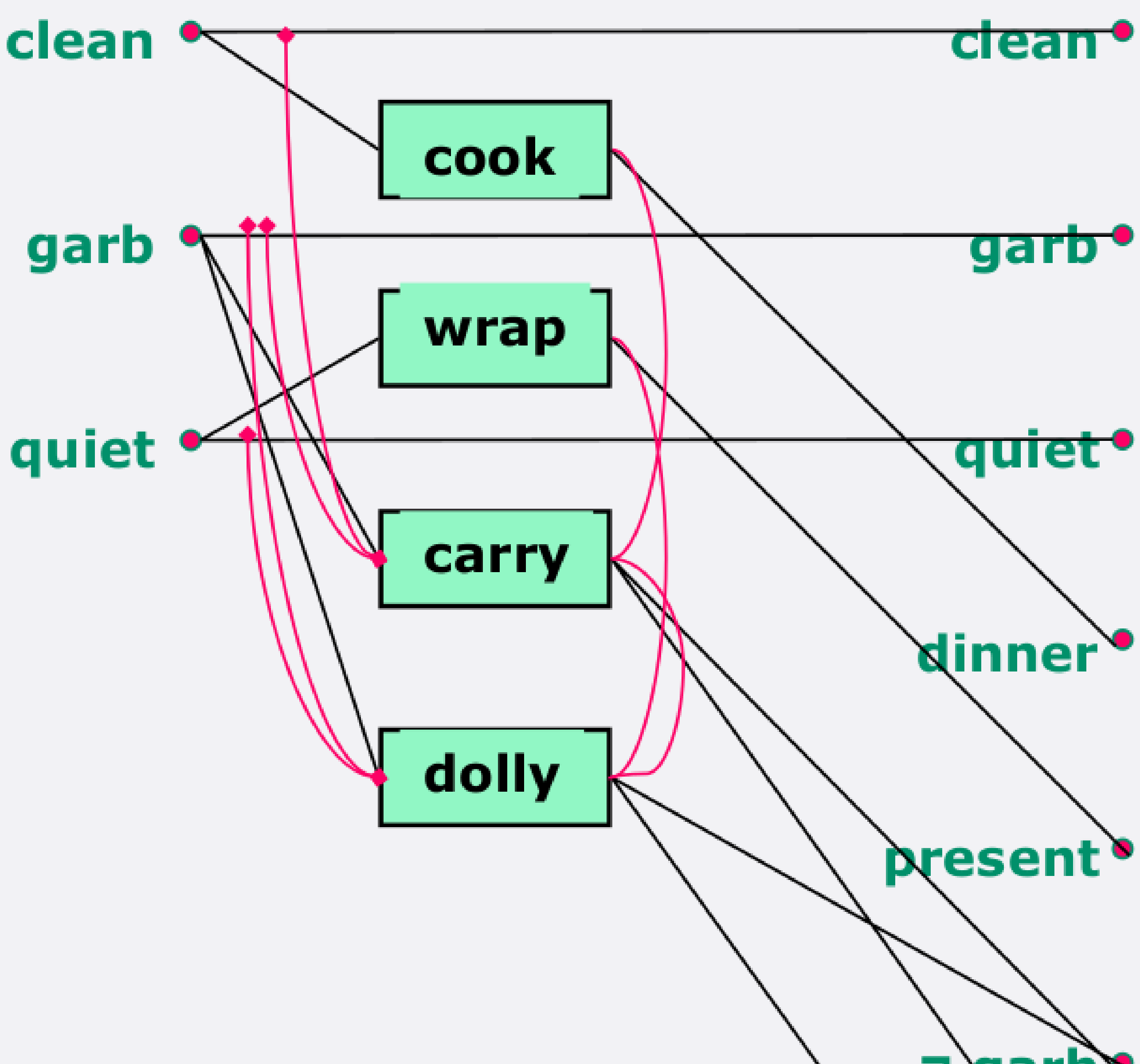
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And we also have interference between **wrap** and **dolly** (dolly makes quiet false, which is required for wrap.)

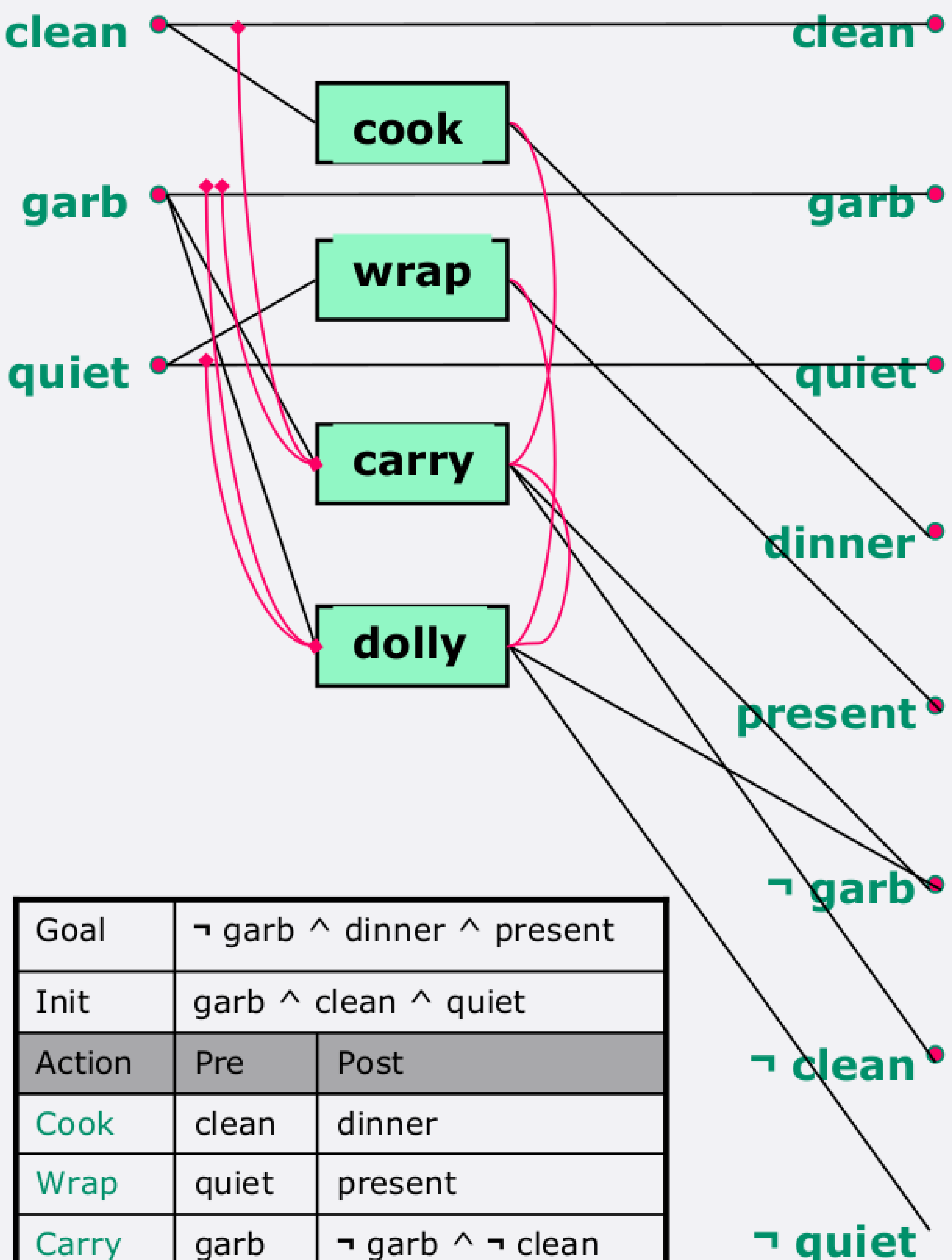
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Finally, we have interference between **carry** and **dolly**, because they each require that garbage be present, and they each remove it. There are two other situations in which we could have action mutexes, but they don’t apply here.

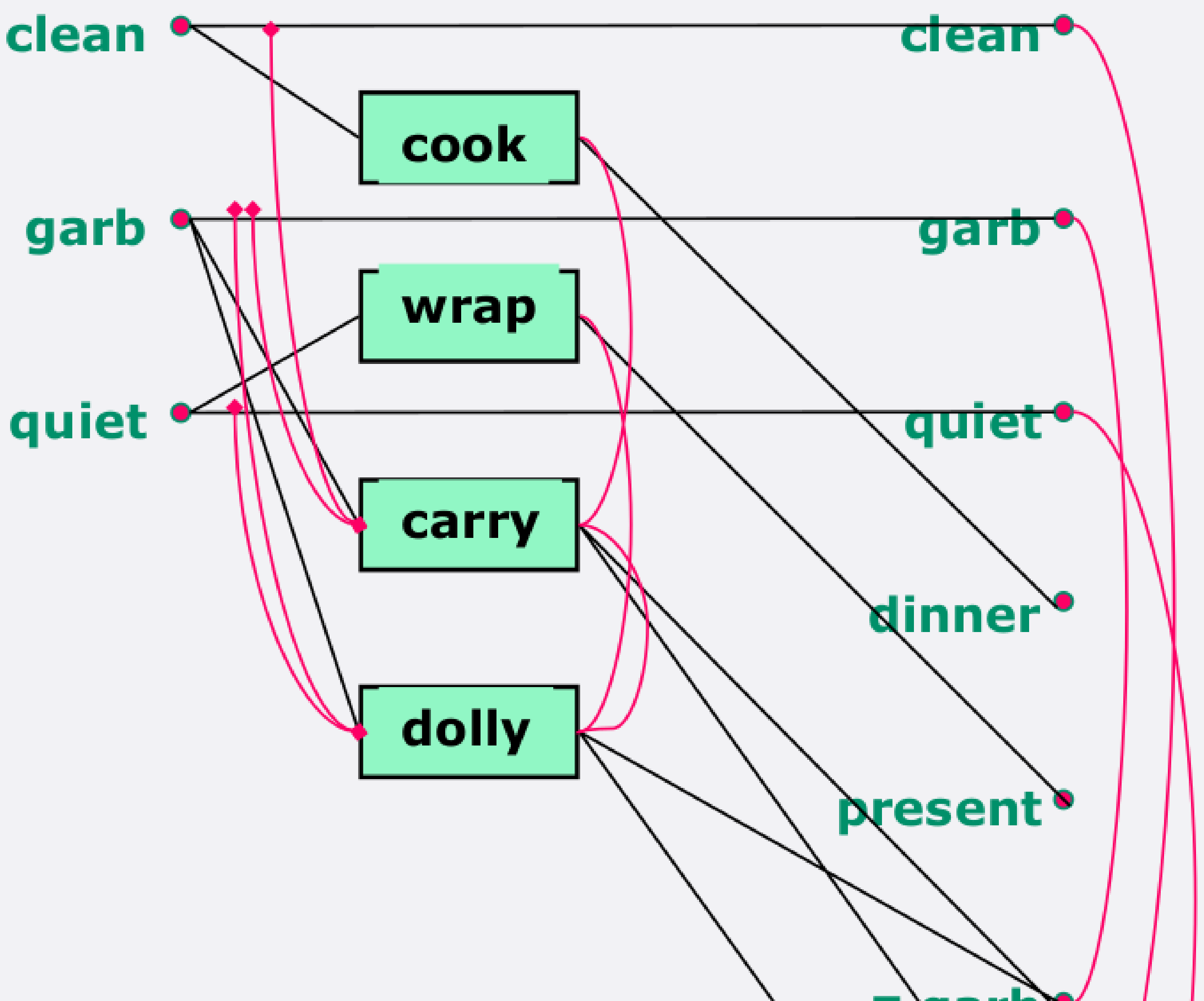
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now let's do the mutexes on the propositions in layer 2.

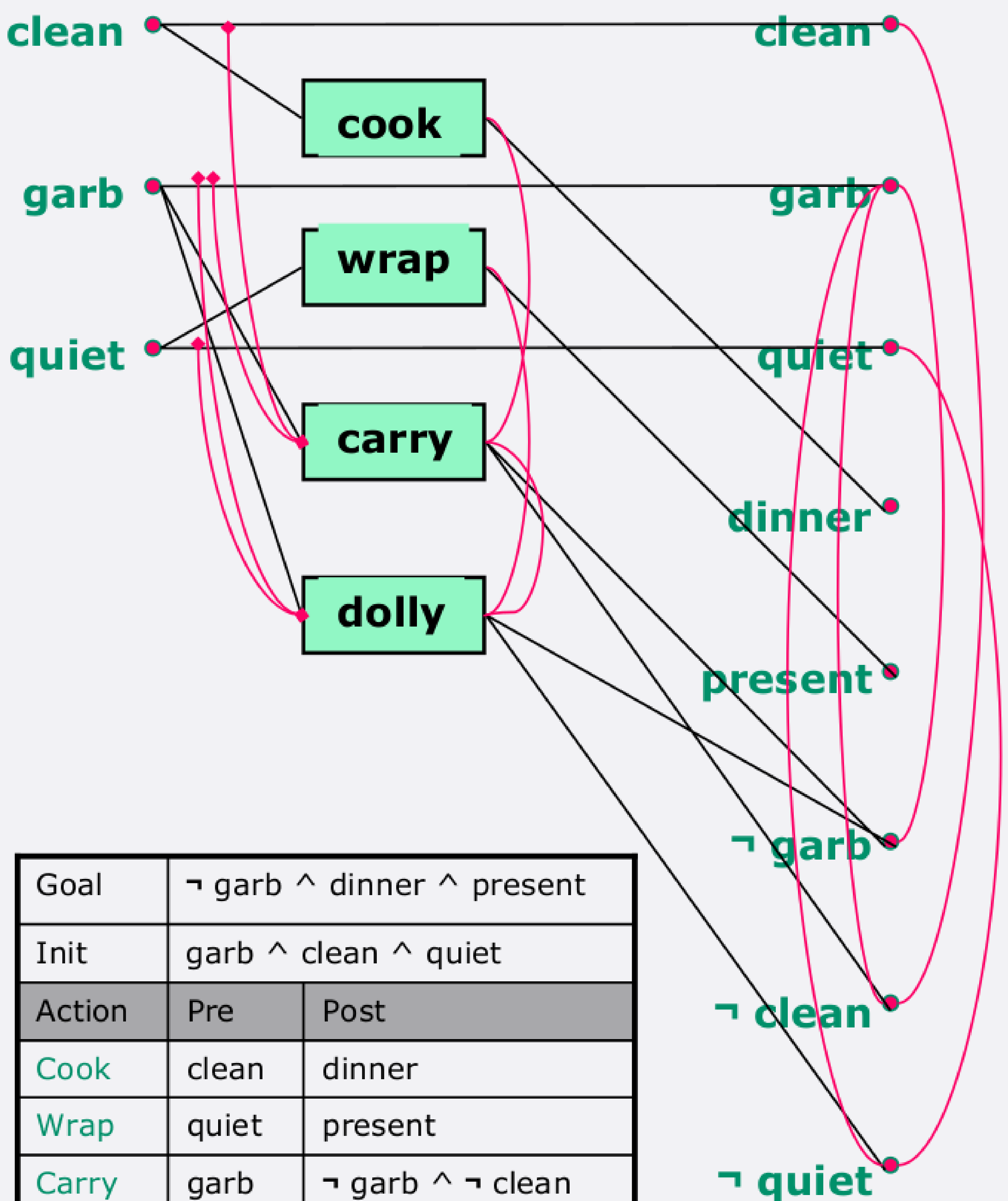
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

First of all, every proposition is mutex with its negation.

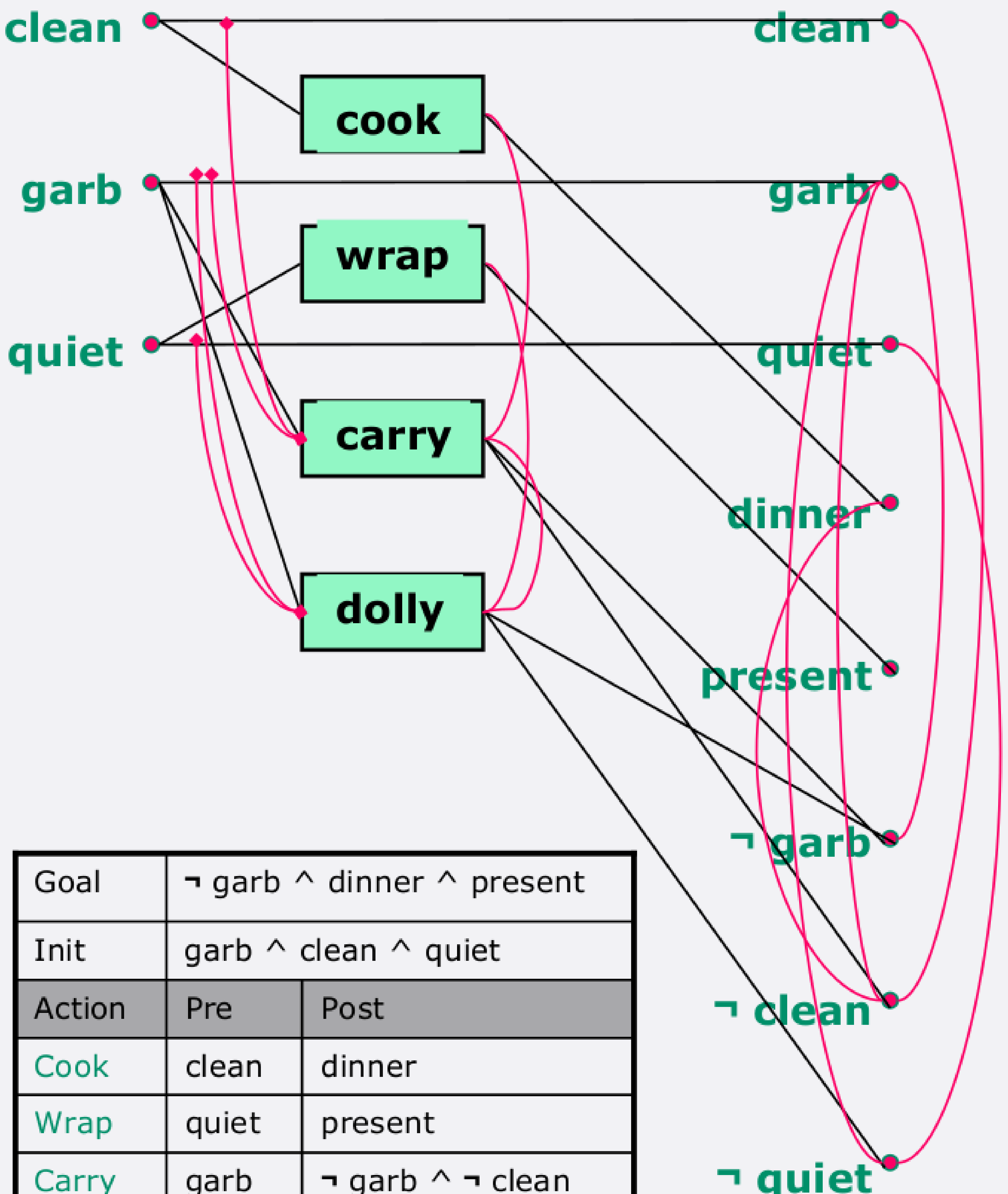
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Then, the other reason we might have mutexes is because of inconsistent support (all ways of achieving the propositions are pairwise mutex). So, here we have that **garbage** is mutex with **not clean** and with **not quiet** (the only way to make garbage true is to maintain it, which is mutex with carry and with dolly).

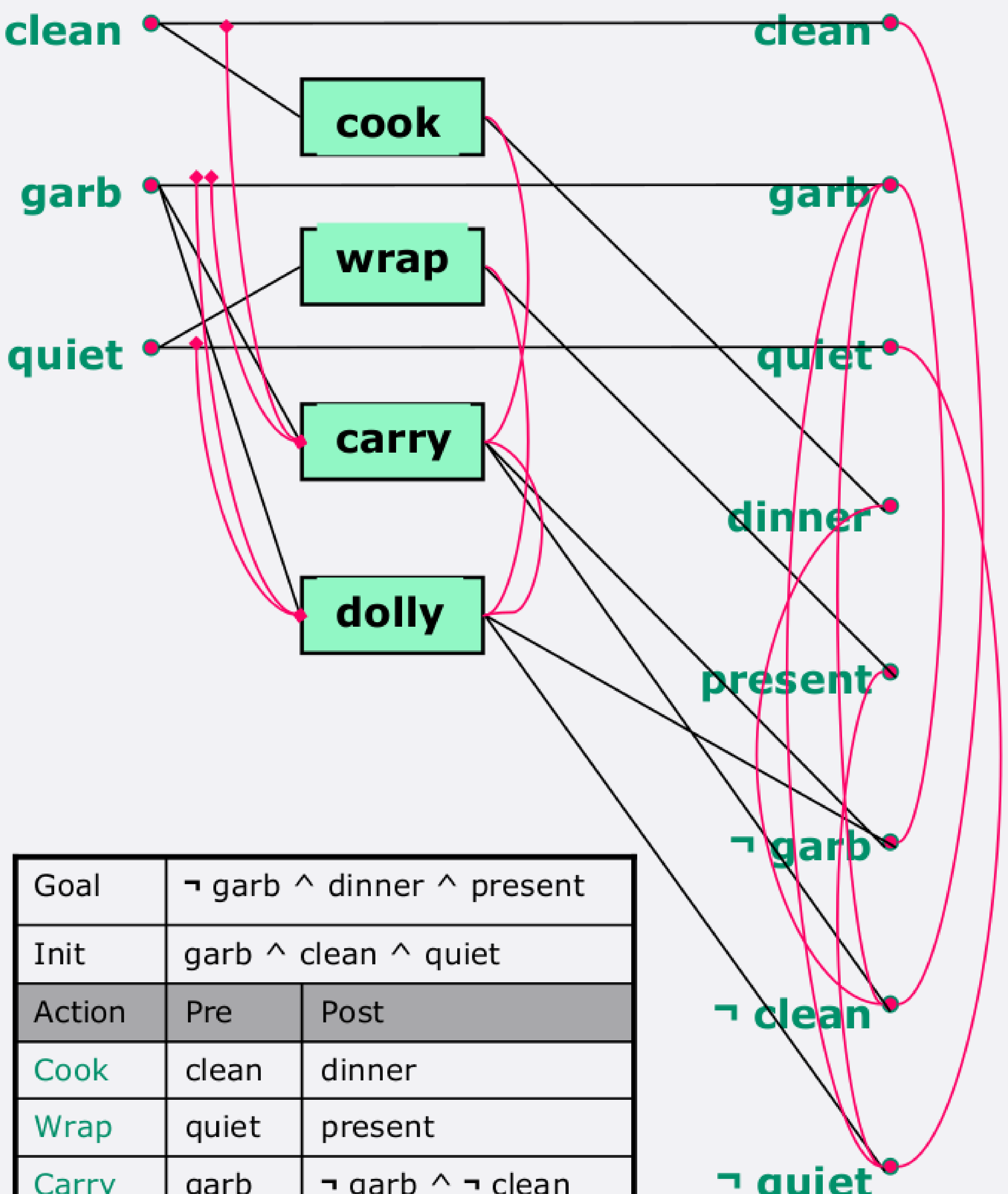
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Dinner is mutex with **not clean** because cook and carry, the only way of achieving these propositions, are mutex at the previous level.

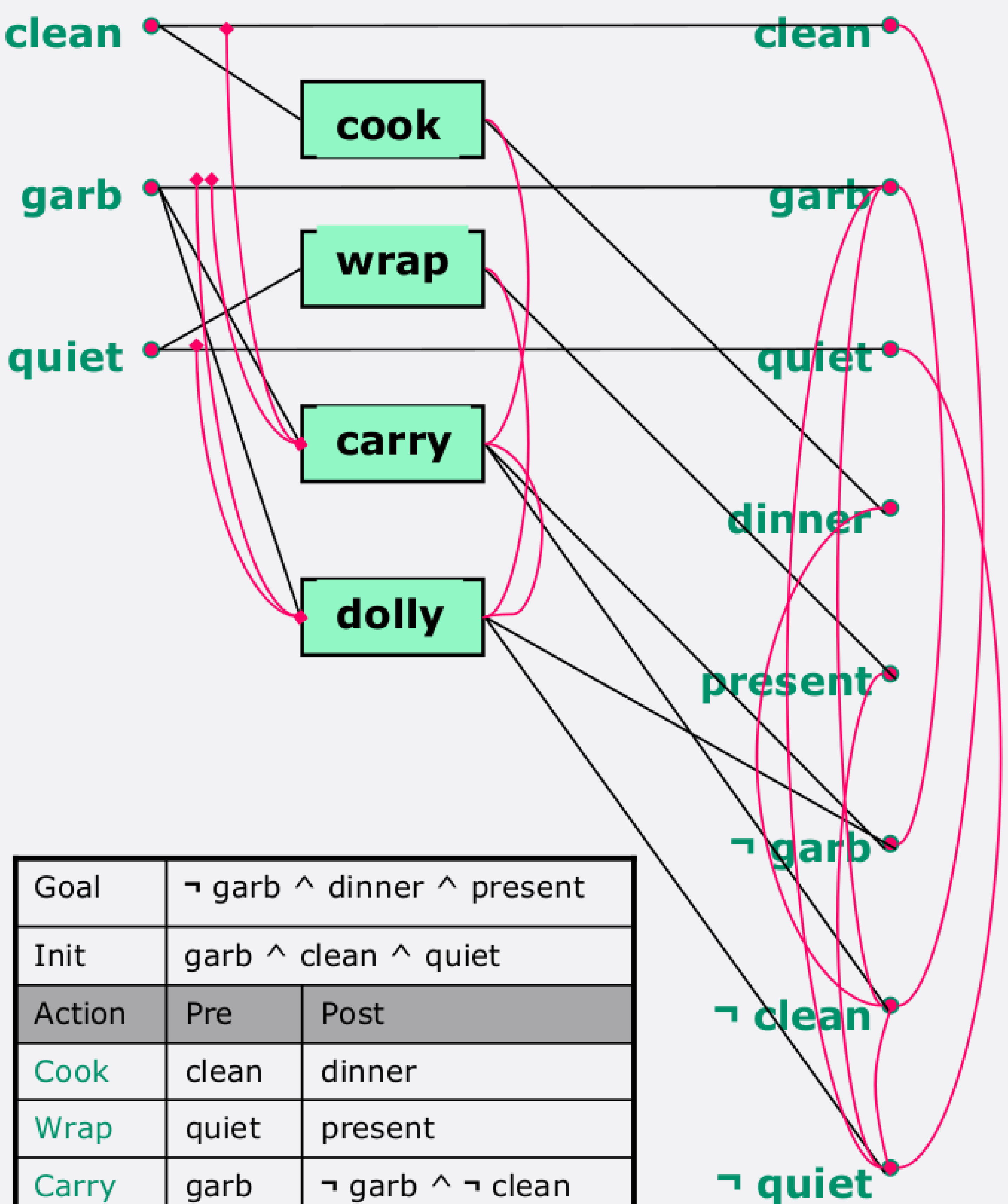
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And **present** is mutex with **not quiet** because wrap and dolly are mutex at the previous level.

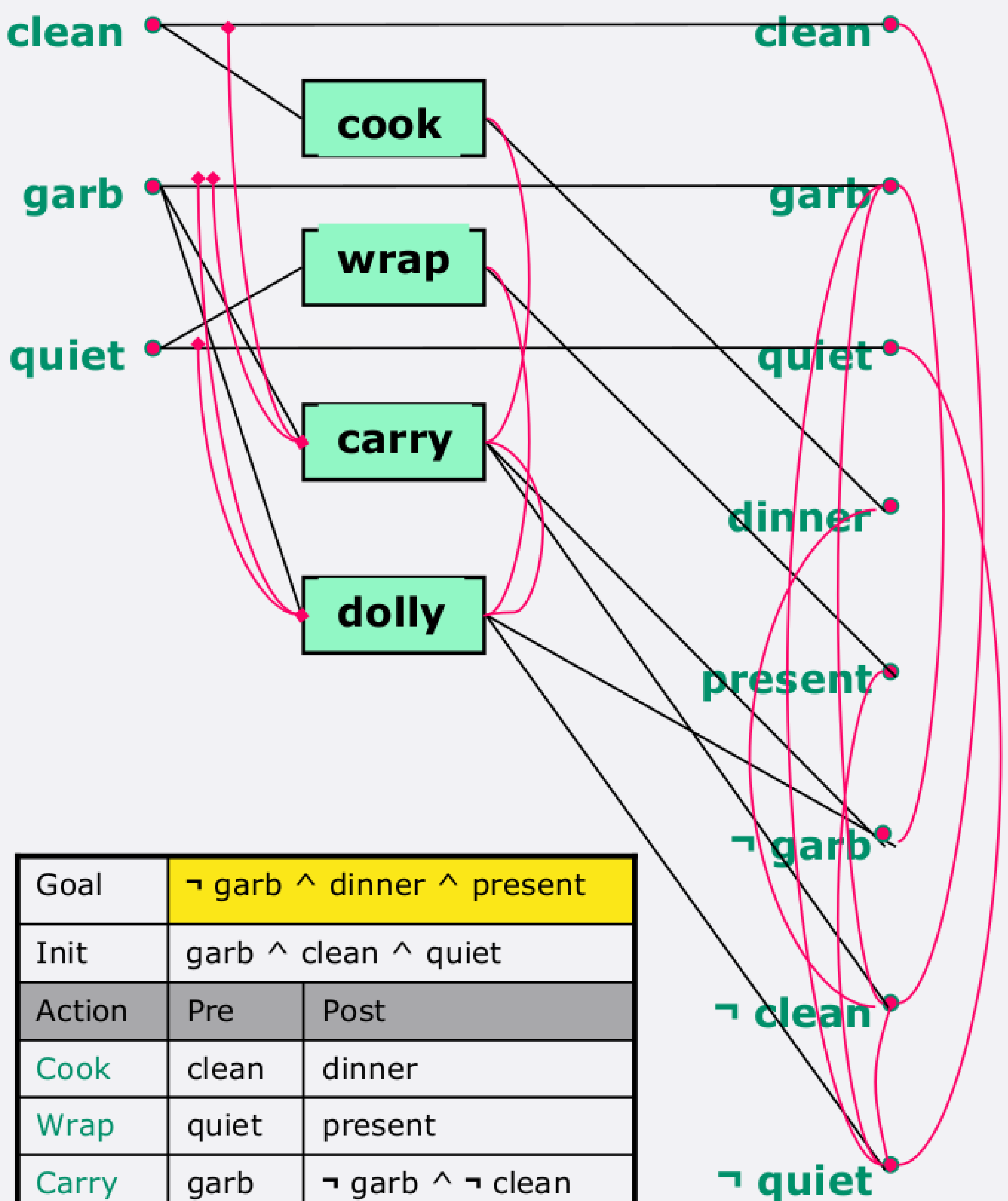
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Finally **not clean** is mutex with **not quiet** because carry and dolly are mutex at the previous level. That's all the mutexes.

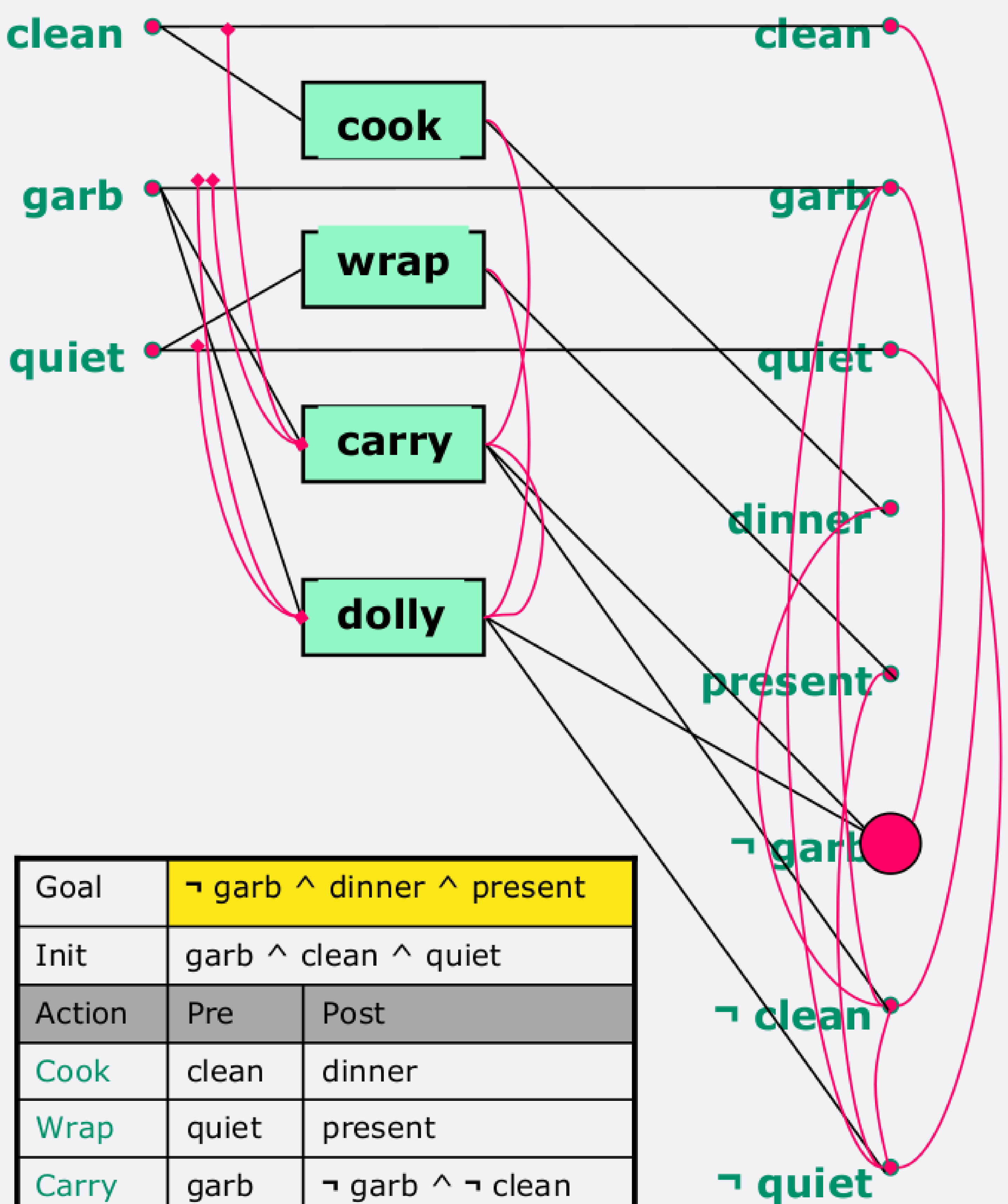
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

So first of all, let's try to ask the question, could the goal conceivably be true?
Our goal is **not garbage** and **dinner** and **present**.
Layer 2 contains not garbage and dinner and present. So it looks like these could possibly be true.
They're not obviously inconsistent.

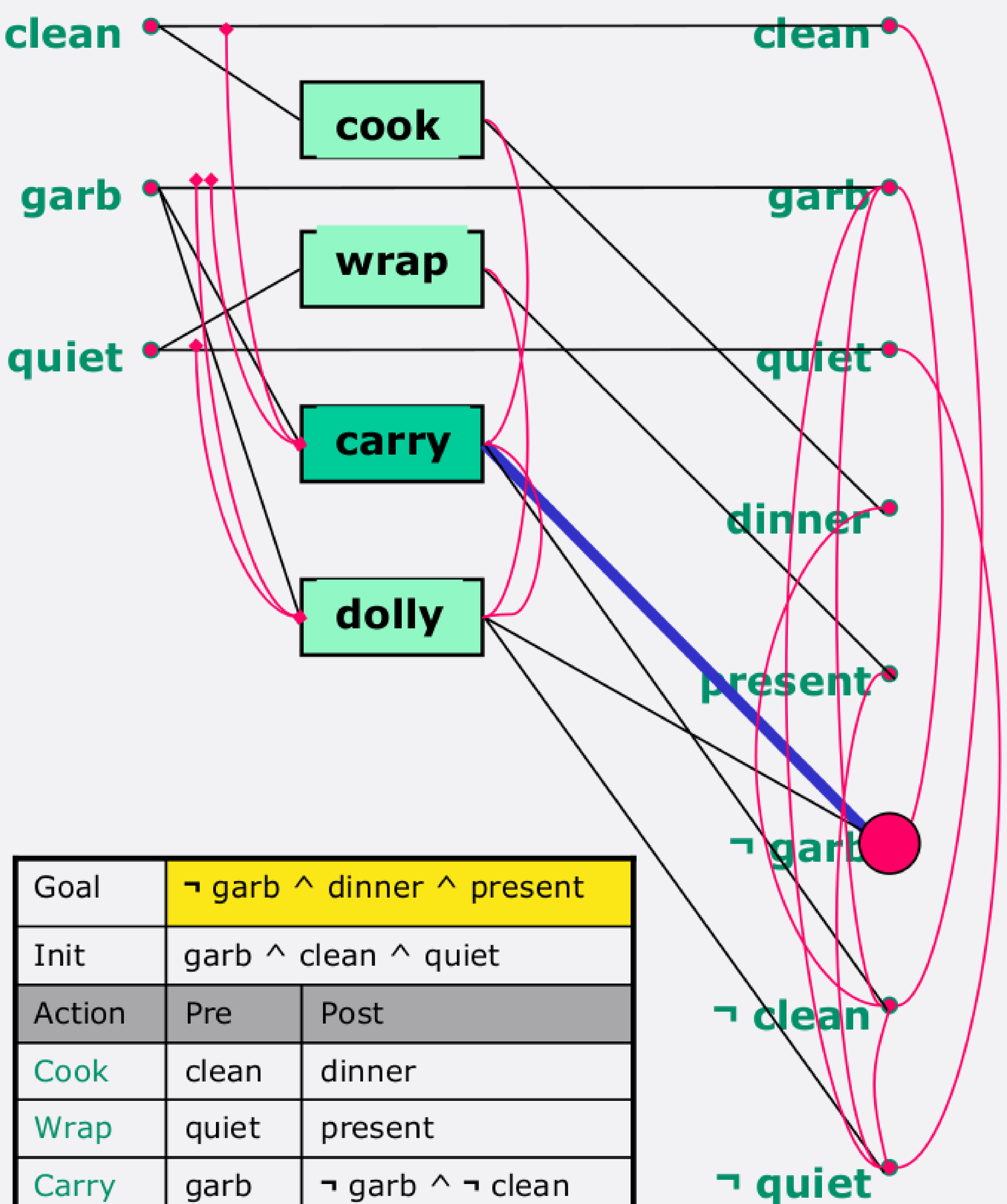
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

So, we'll start looking for a plan by finding a way to make not garbage true.

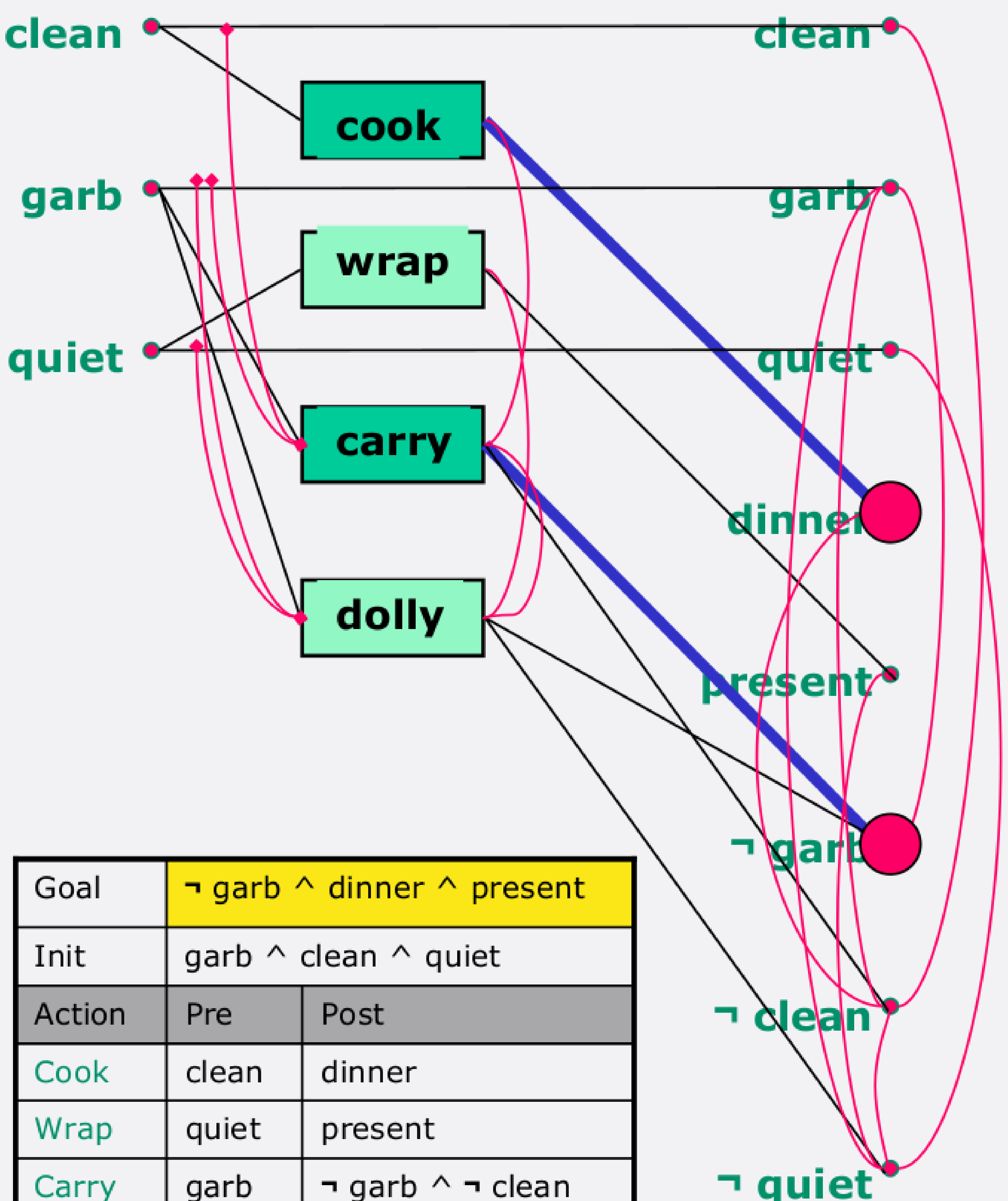
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We'll try using the carry action.

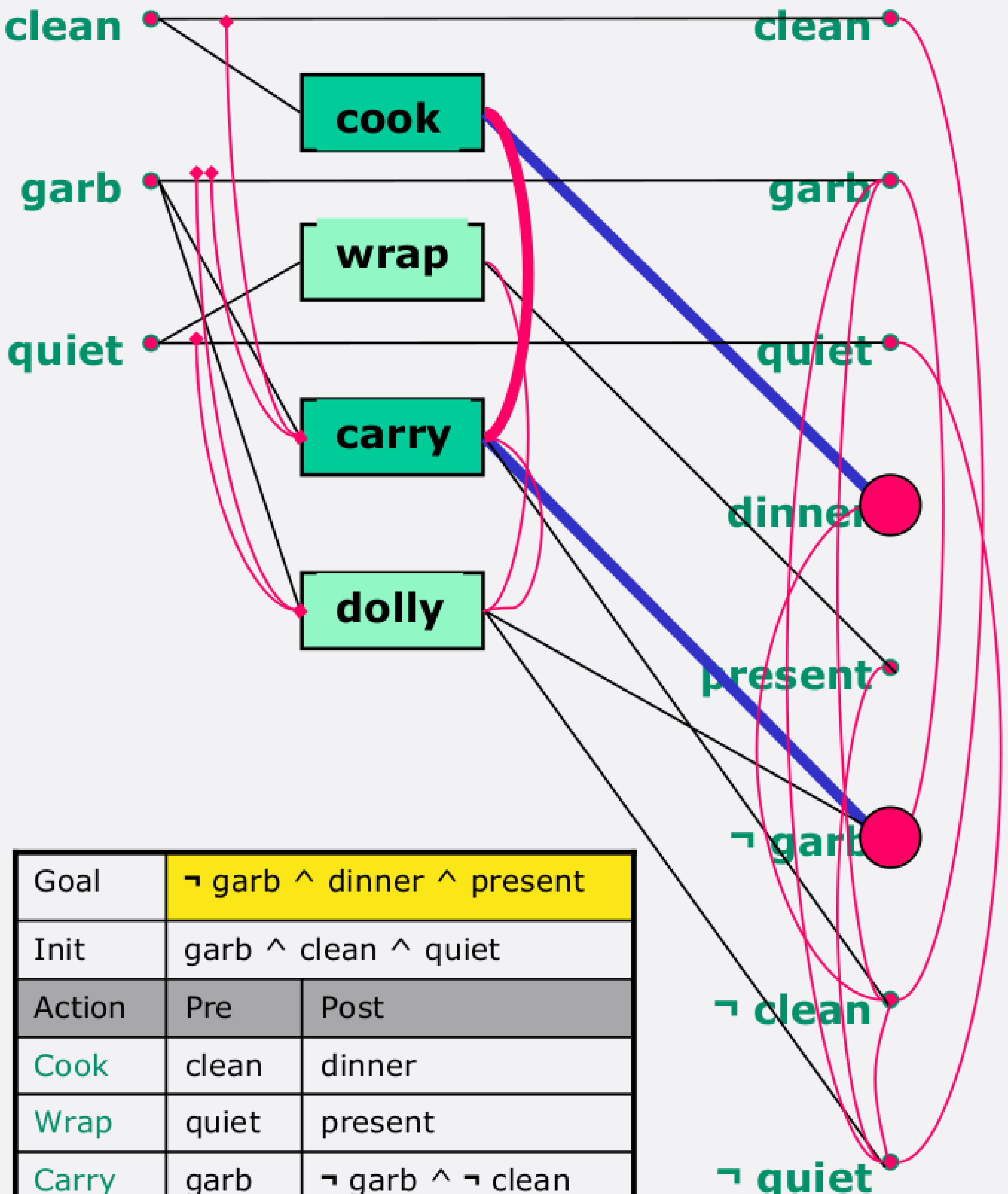
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now, we'll try to make dinner true the only way we can, with the cook action.

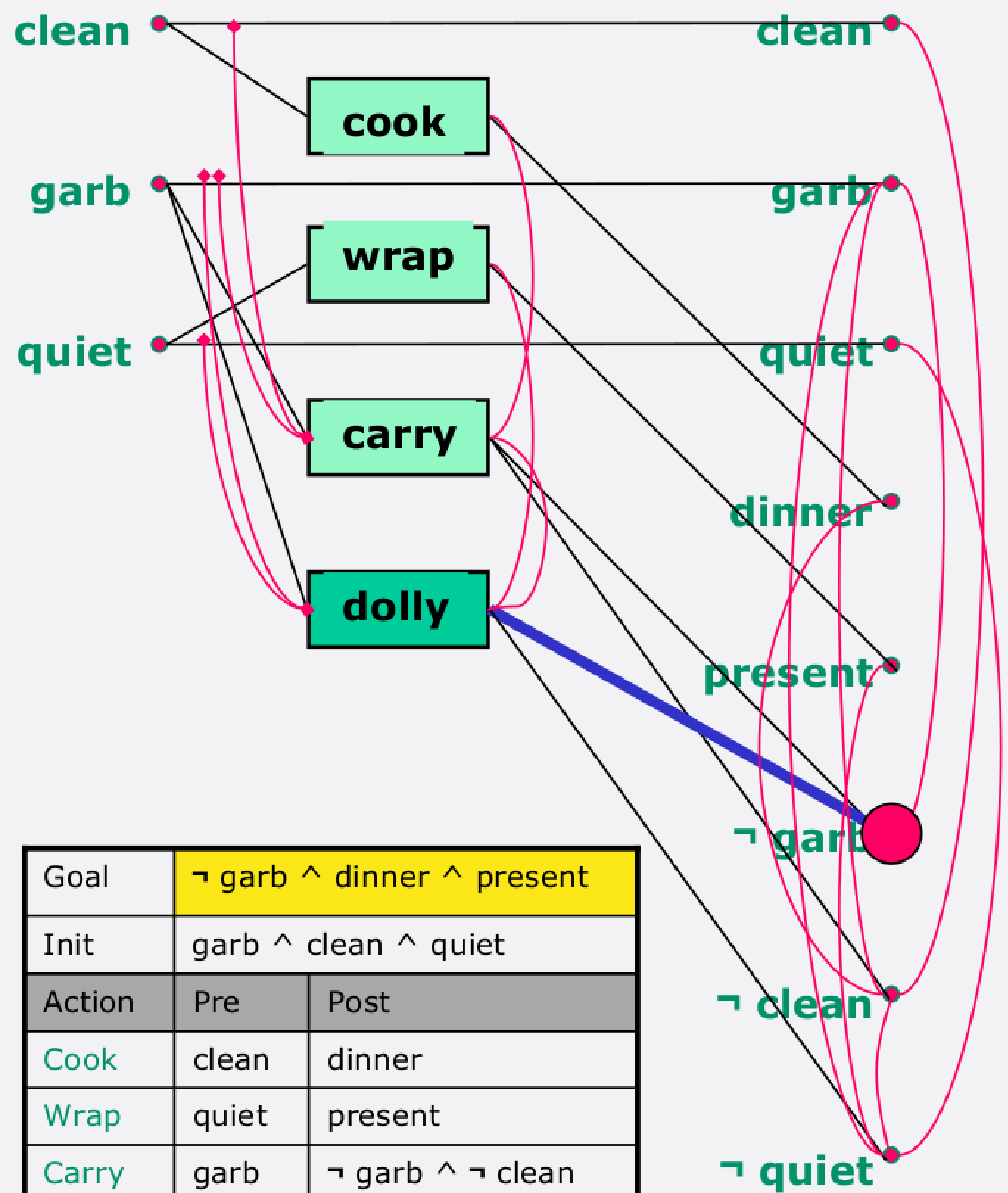
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

But cook and carry are mutex, so this won’t work.

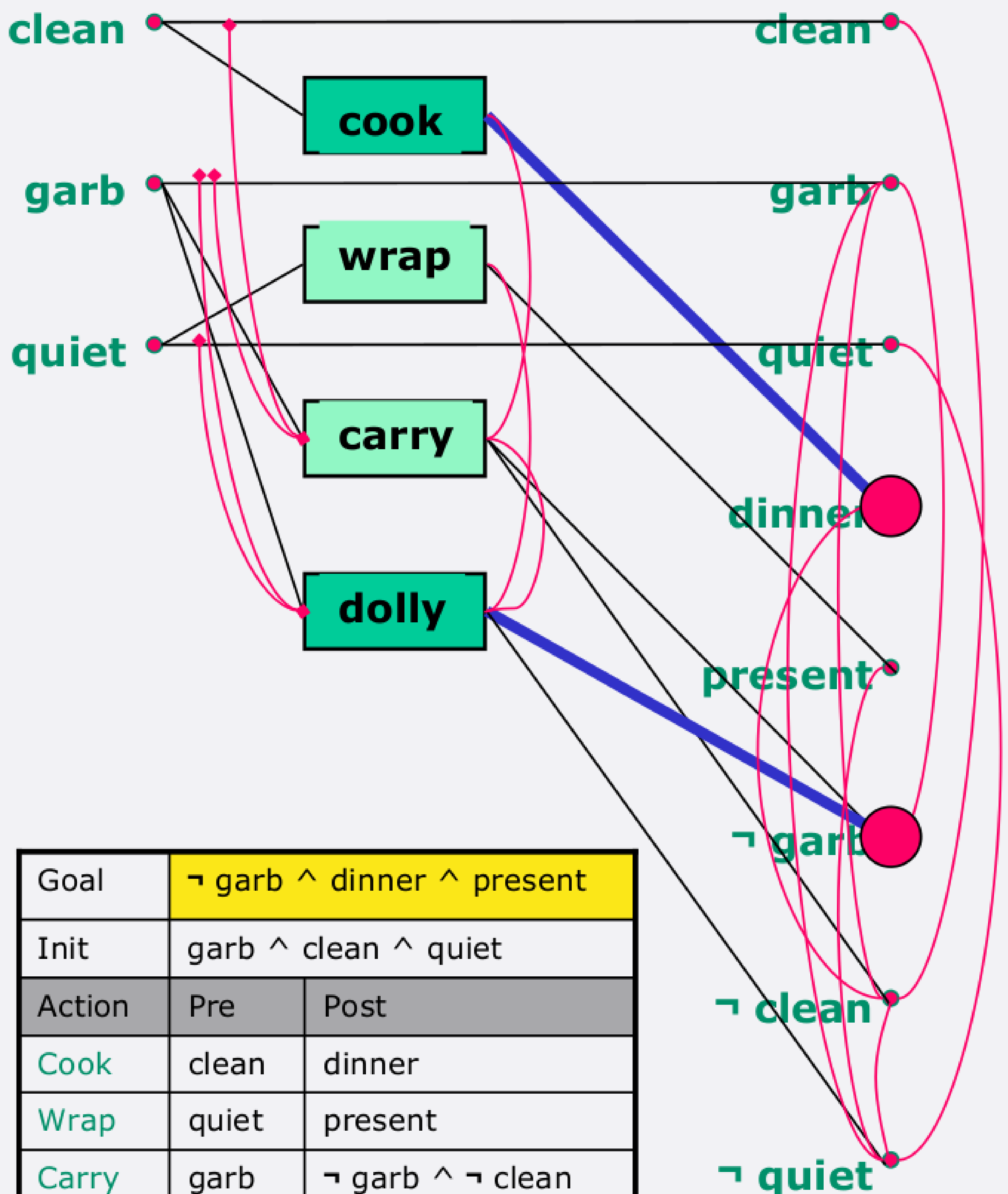
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Because there aren't any other ways to make dinner, we fail, and have to try a different way of making not garbage true. This time, we'll try dolly.

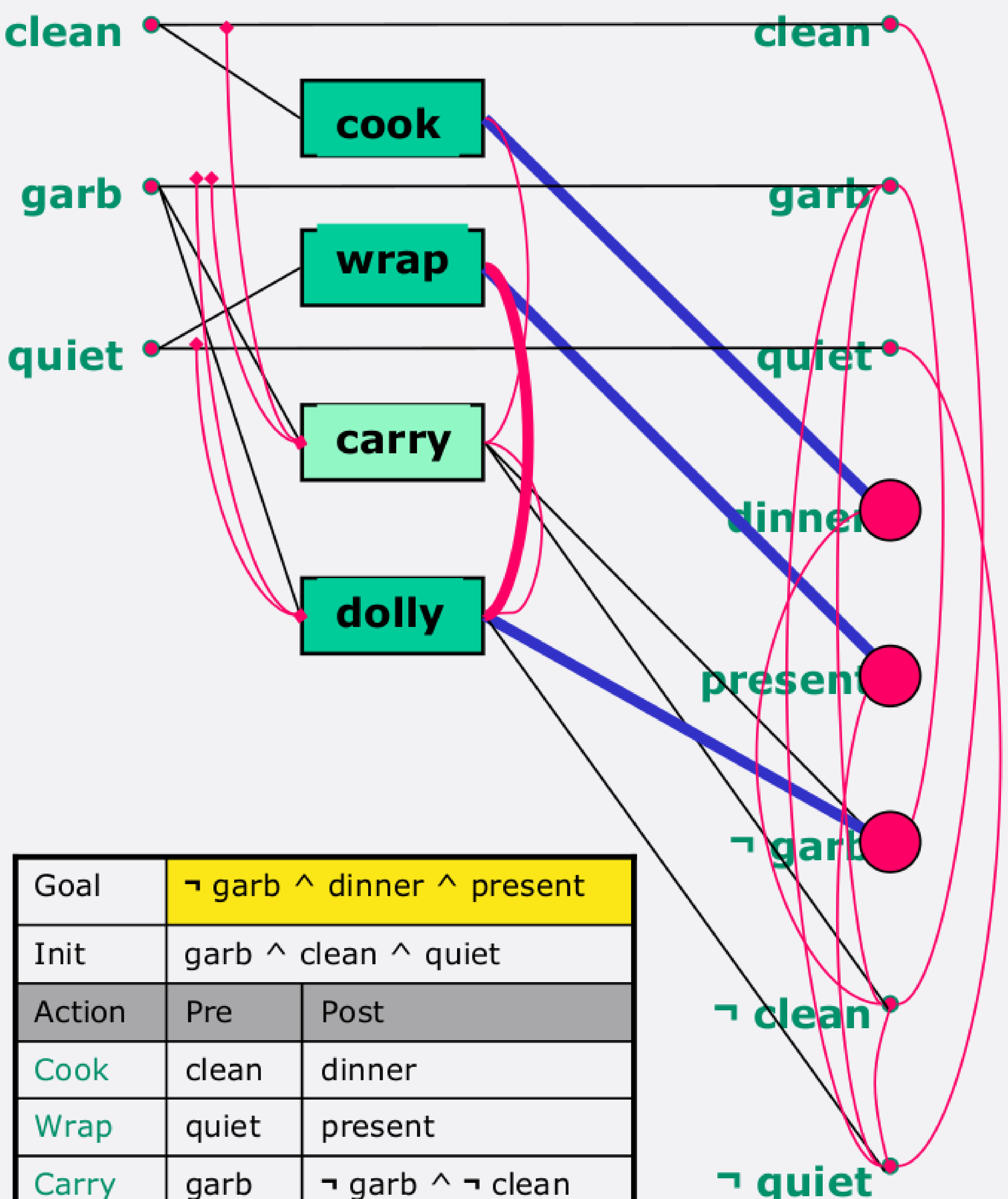
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now, we can cook dinner, and we don't have any mutex problems with dolly.

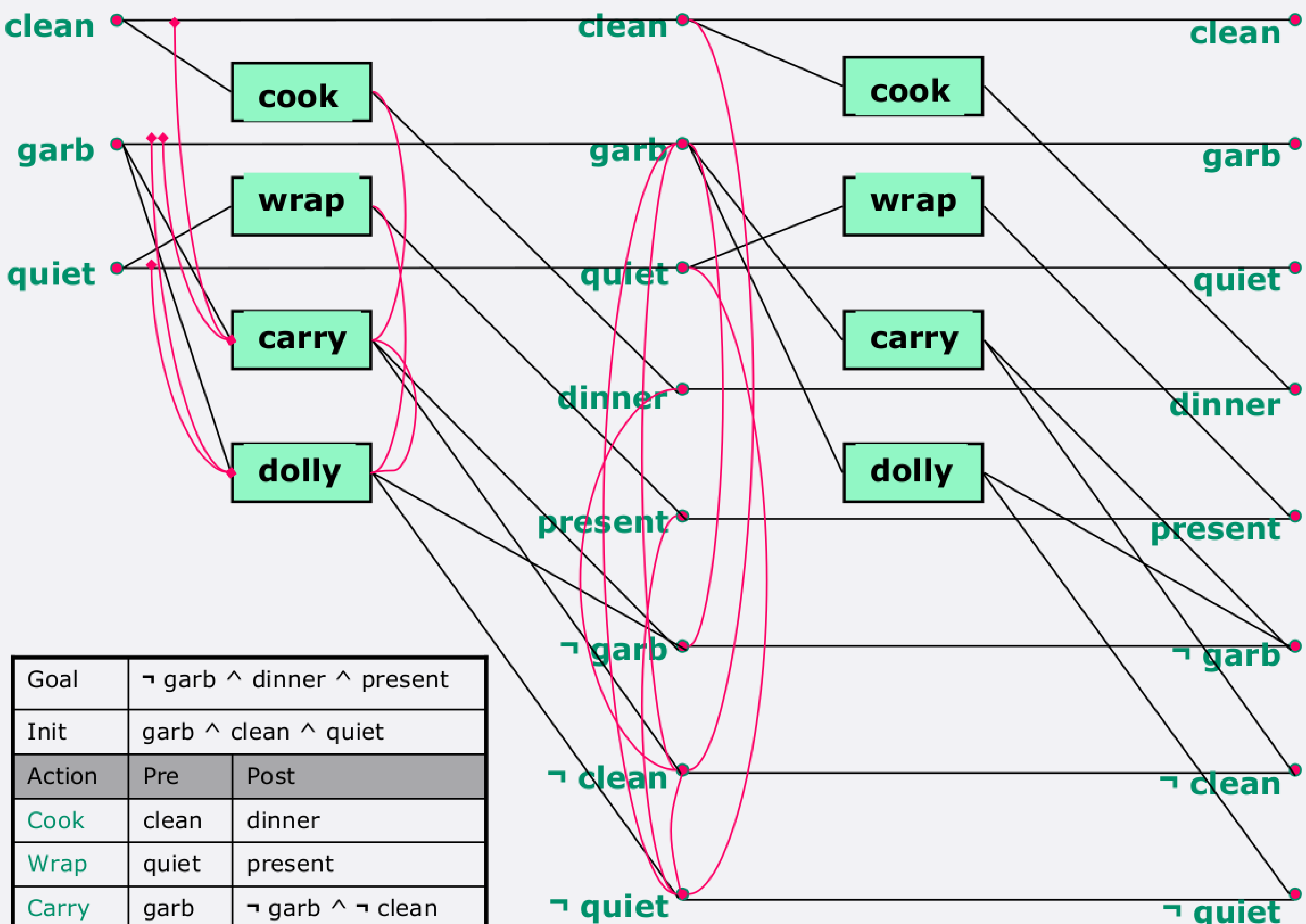
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We have to make present true as well. The only way of doing that is with wrap, but wrap is mutex with dolly. So, we fail completely.

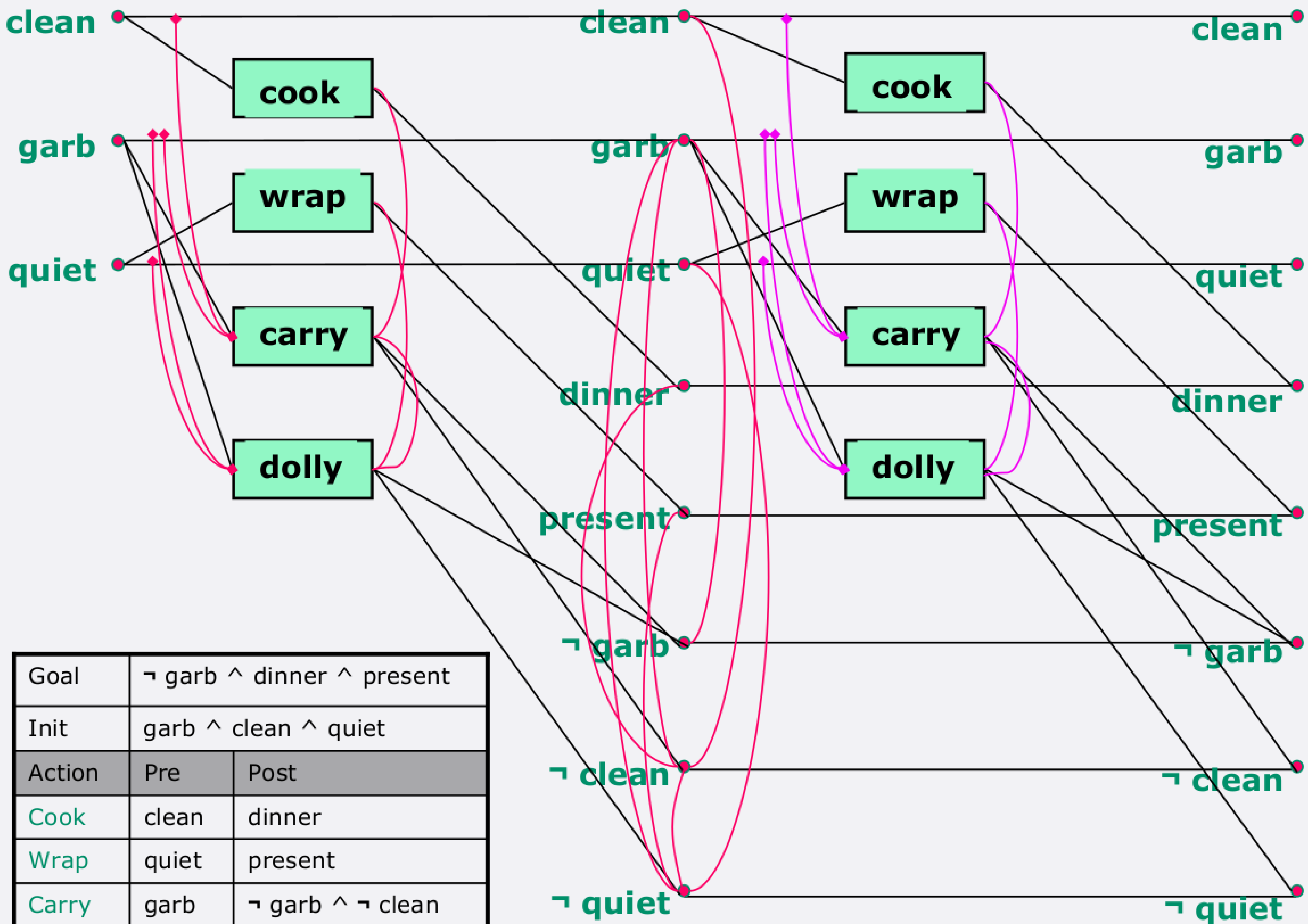
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	**Pre**	**Post**
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

There's no way to achieve all of these goals in parallel. So we have to consider a depth two plan.
We start by adding another layer to the plan graph.

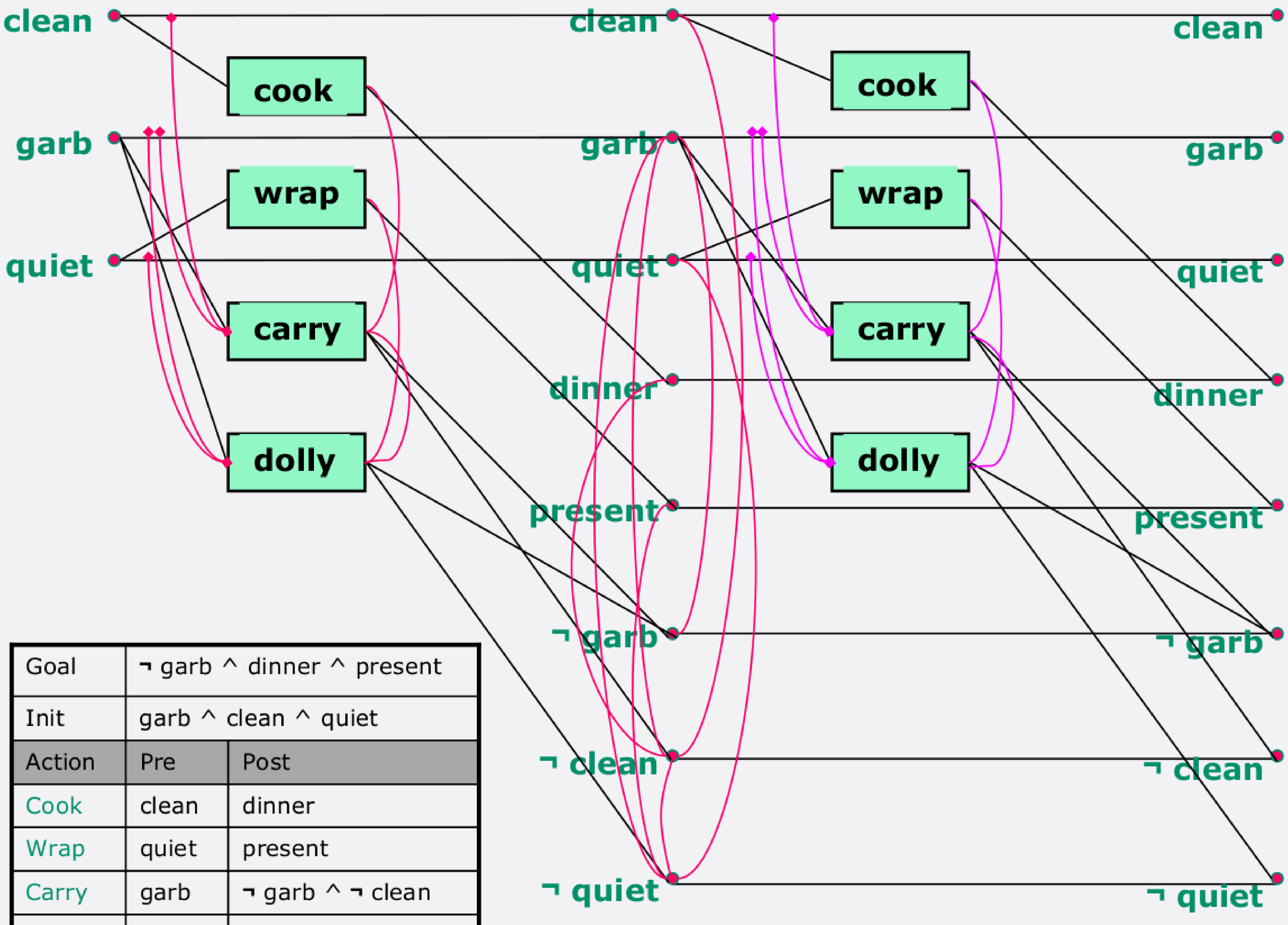
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We have the same set of mutexes on actions that we had before.

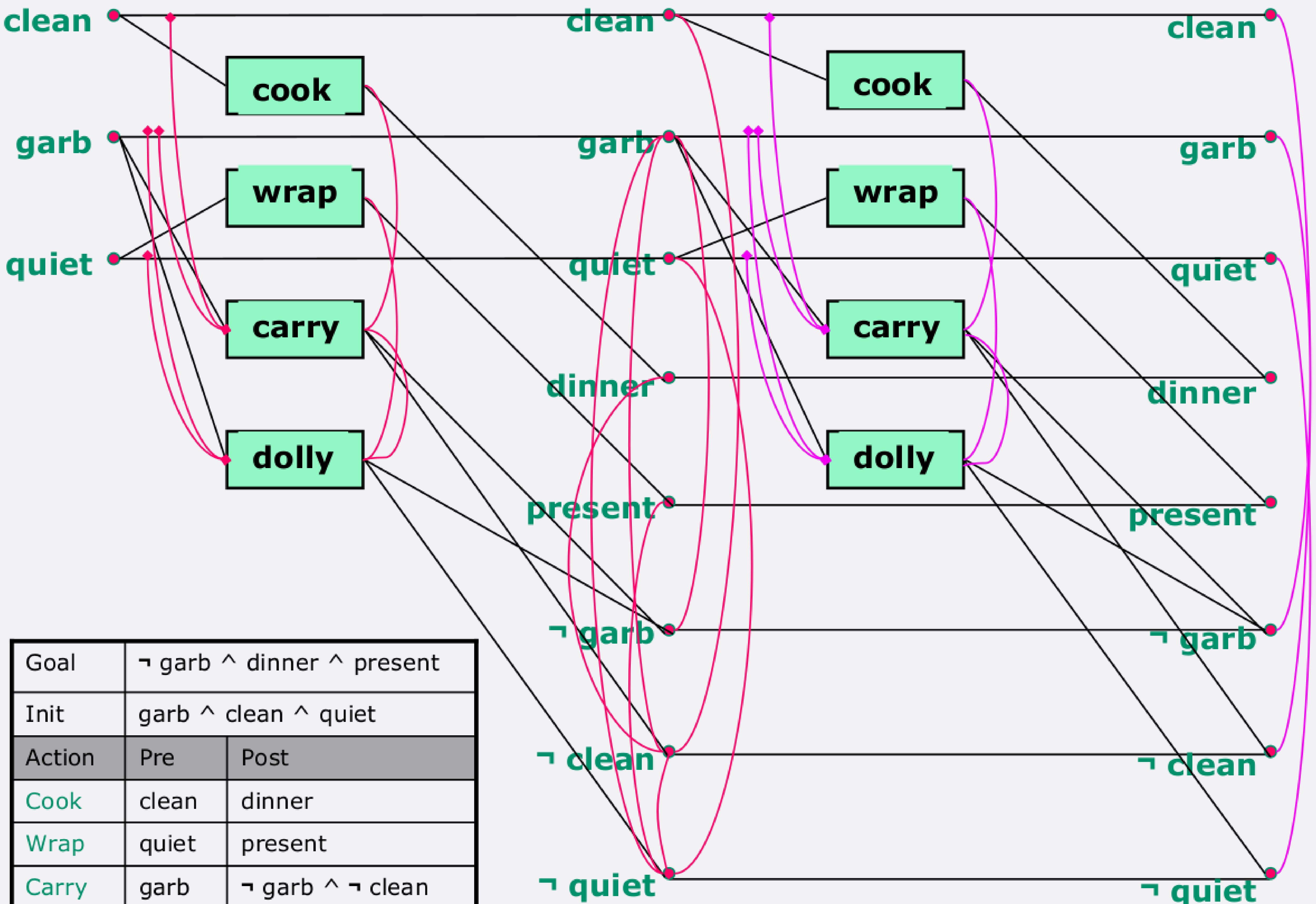
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

There is also a large set of additional mutexes between maintenance actions for not garbage, not clean, and not quiet. I'm going to leave them out of this graph, in the interests of making it readable (and they're not going to affect the planning process in this example).

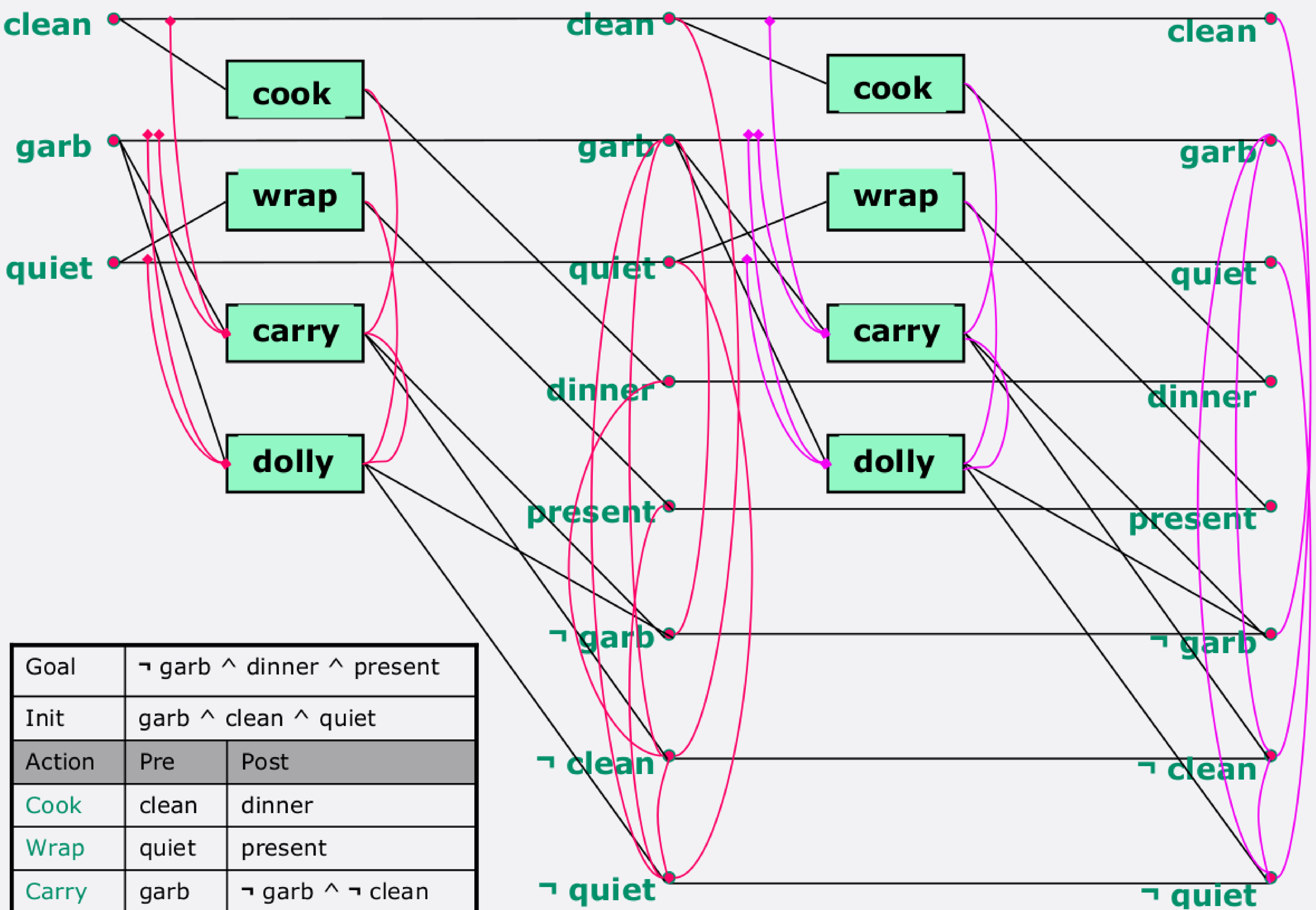
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

So let's look at the proposition mutexes in layer 4. We still have that every proposition is mutex with its negation.

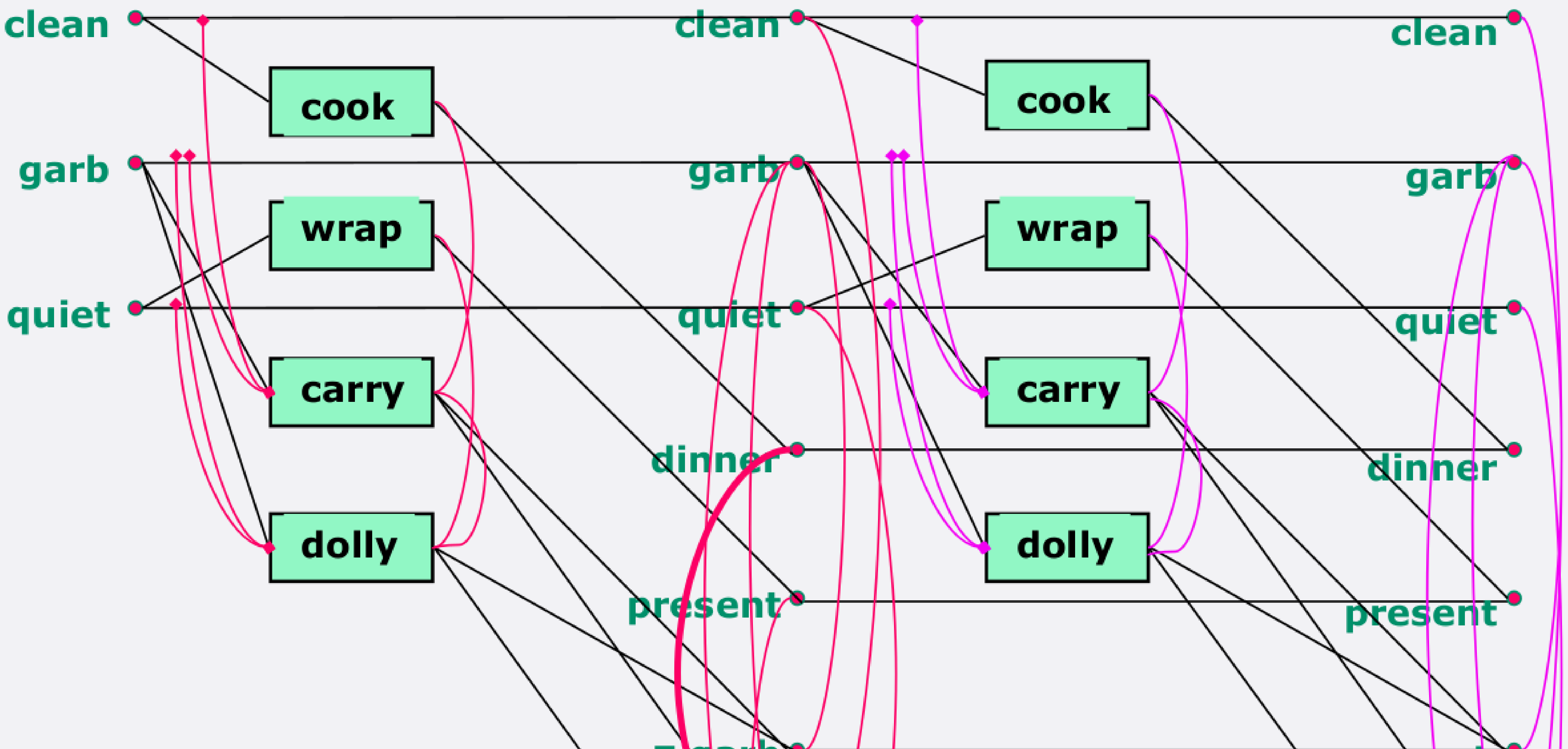
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And we get some of the same mutexes that we had in the previous proposition layer.

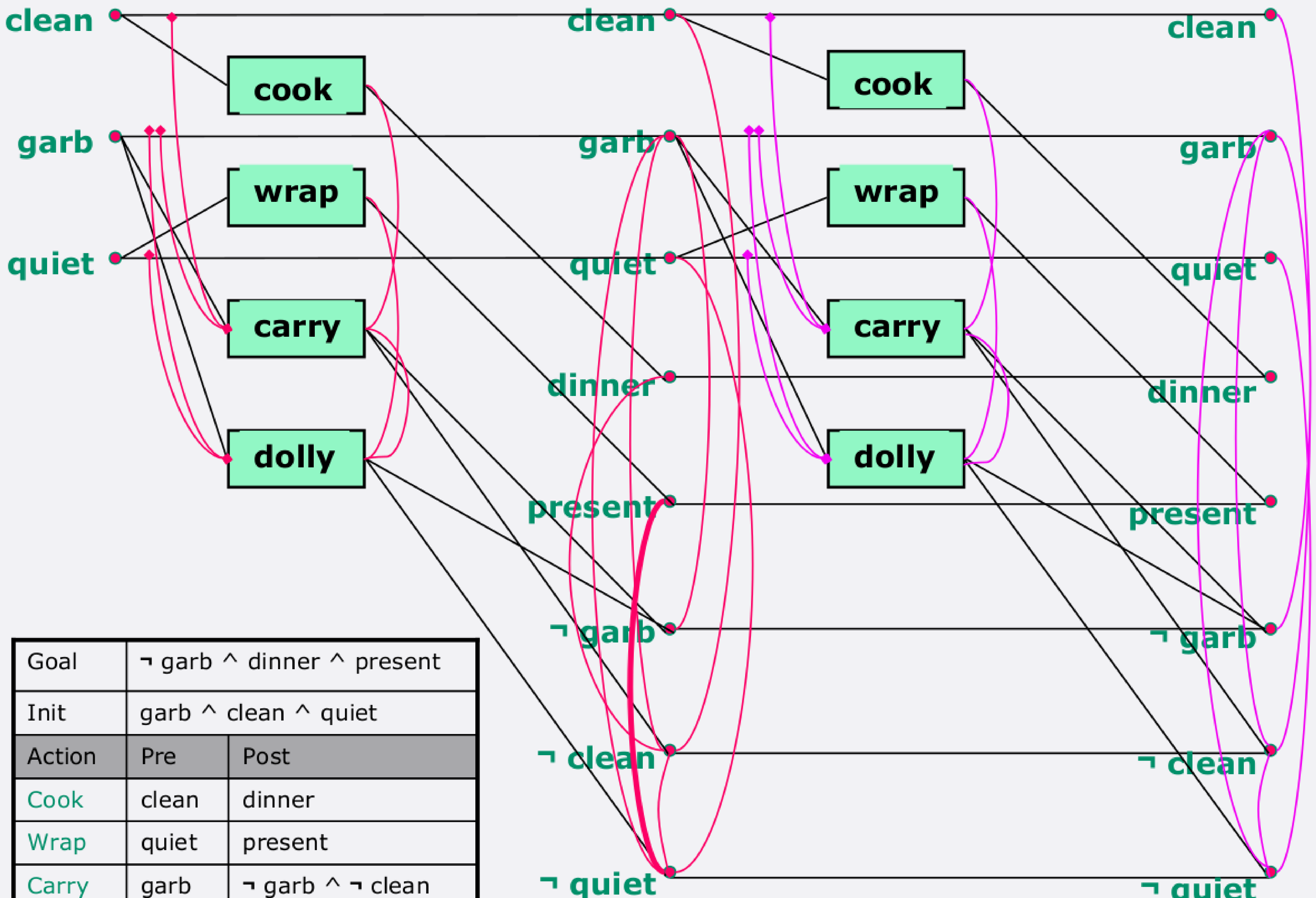
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

In layer 2, we had a mutex between **dinner** and **not clean**. But we don't have it in layer 4, because it's possible to make dinner true by maintaining it, and making not clean true by carry. And those two actions are consistent with one another at level 3.

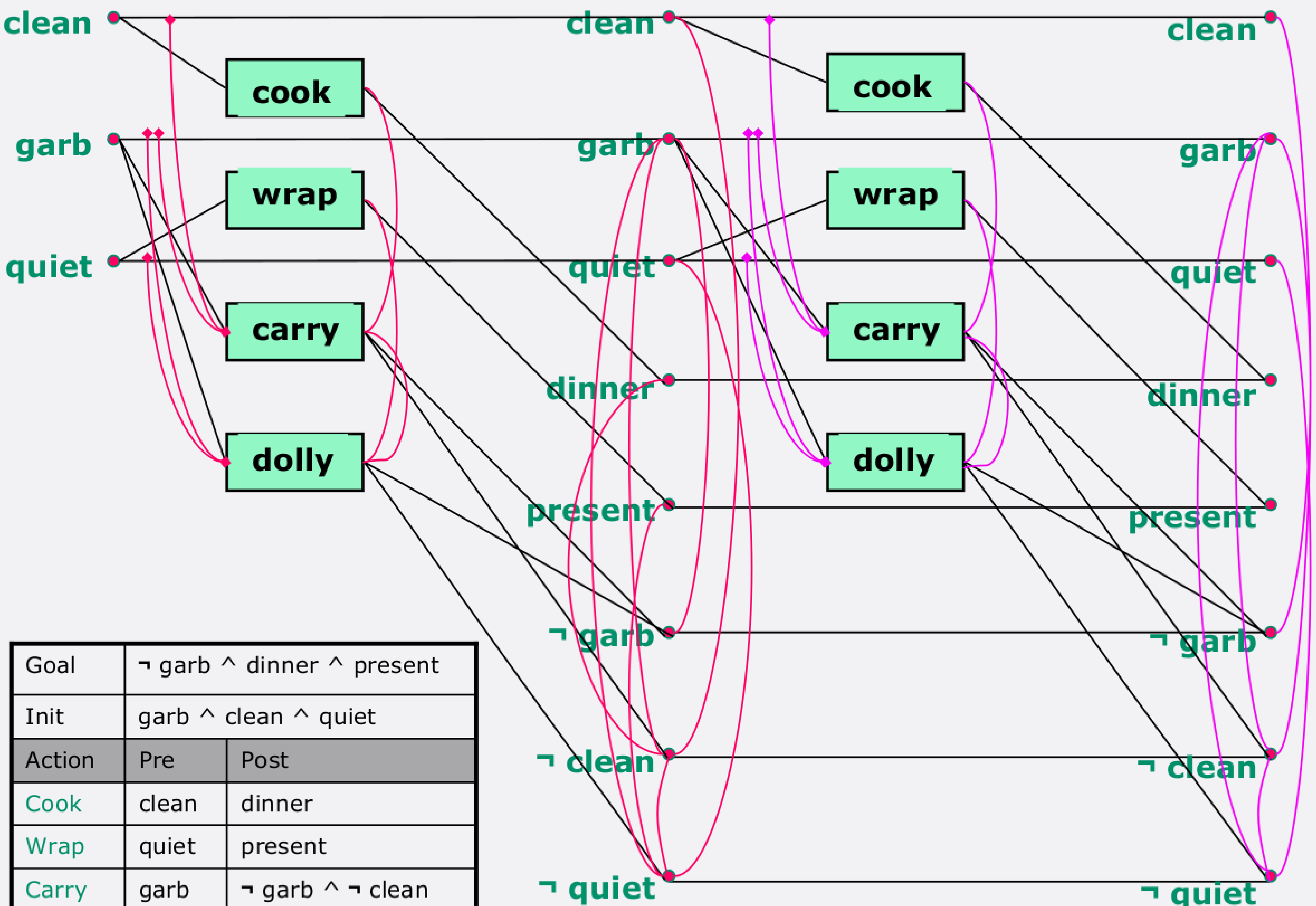
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Similarly, in layer 2 we had a mutex between **present** and **not quiet**. But we don't have it here because we can make present true by maintaining it and make not quiet true by dolly.

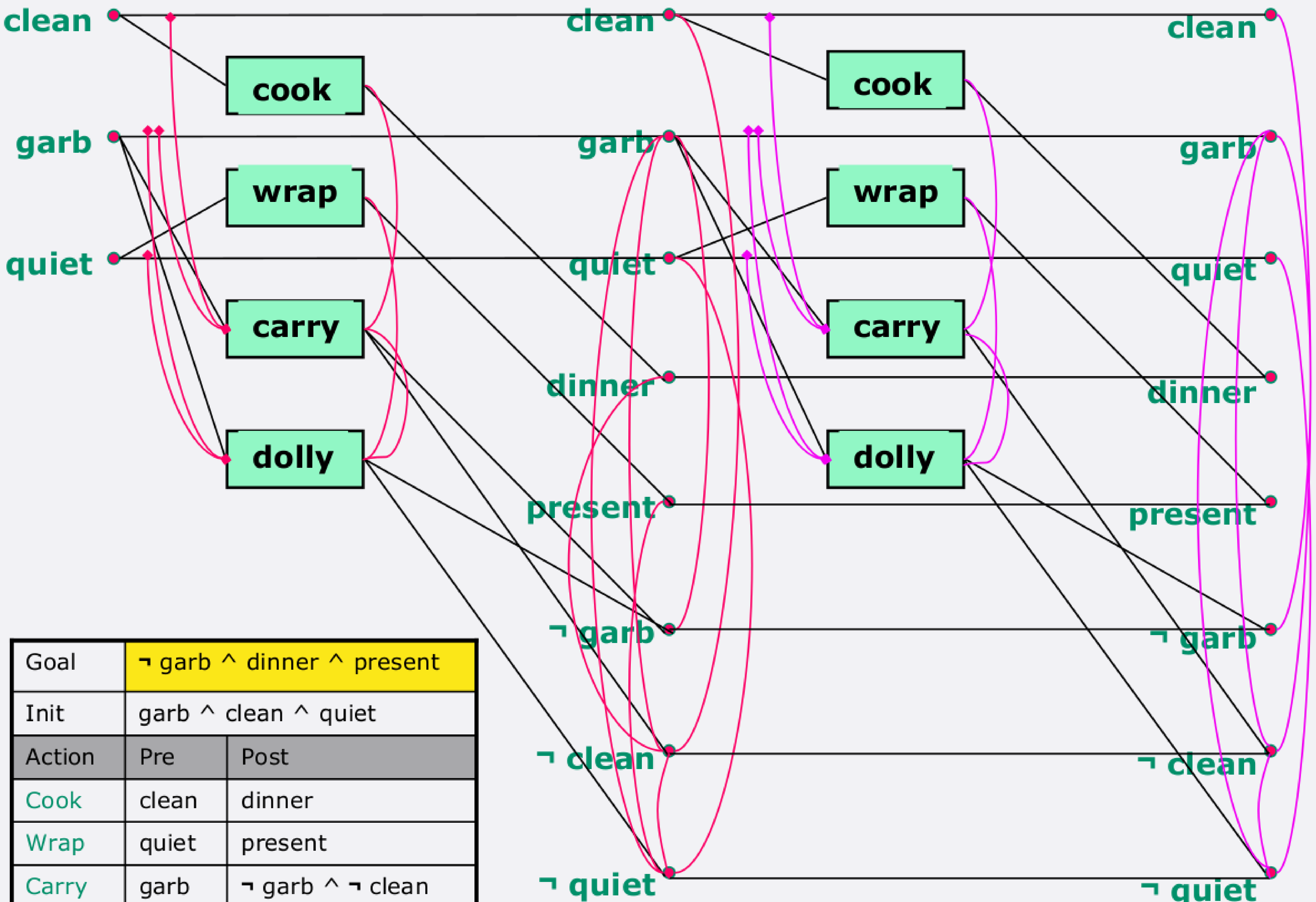
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

It’s important to see that, by giving ourselves an added time step, there are fewer mutexes, and so more things we can accomplish.

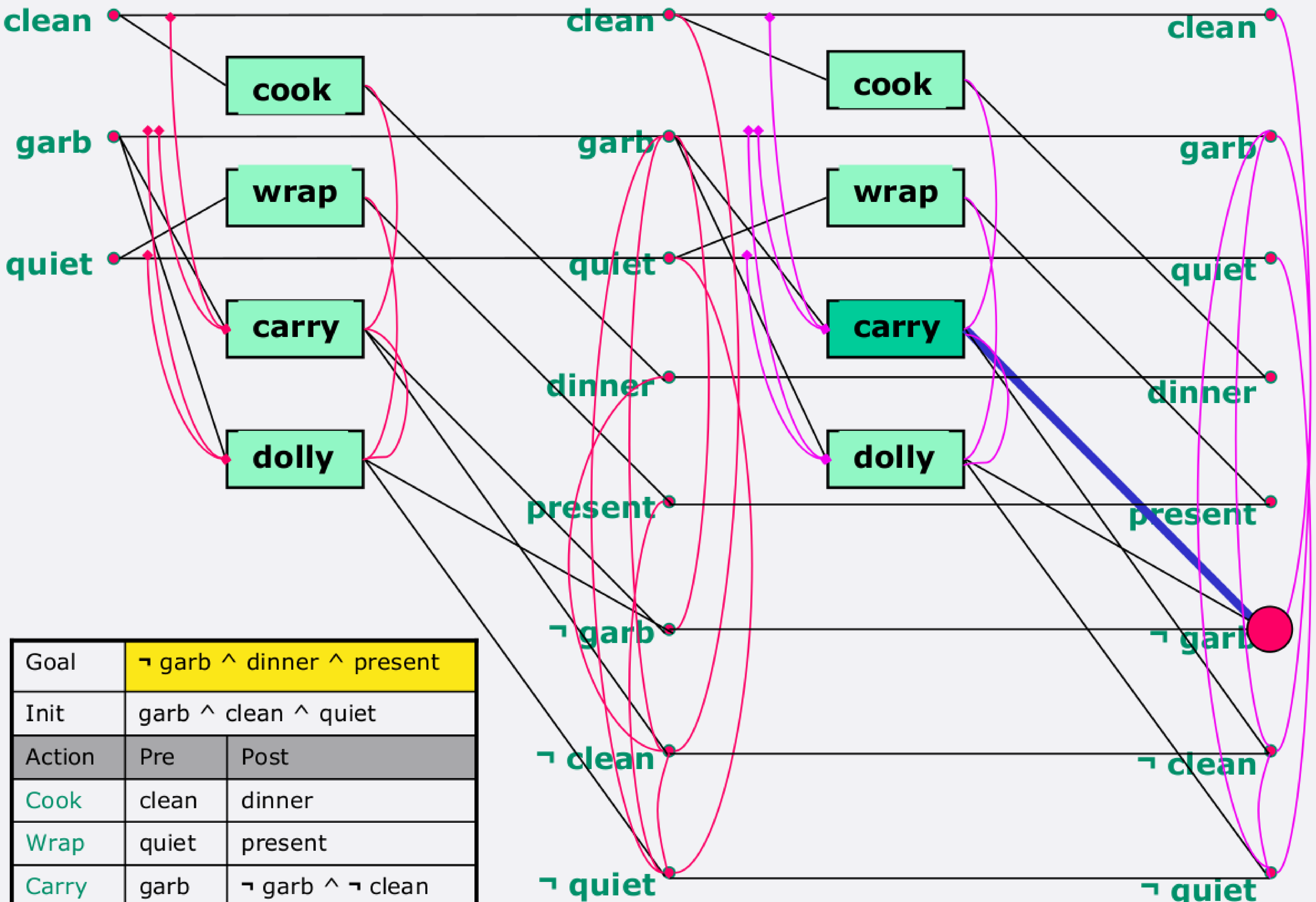
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now it’s time to try to find a plan again. All of our goal conditions are present in the last layer, so let’s start searching.

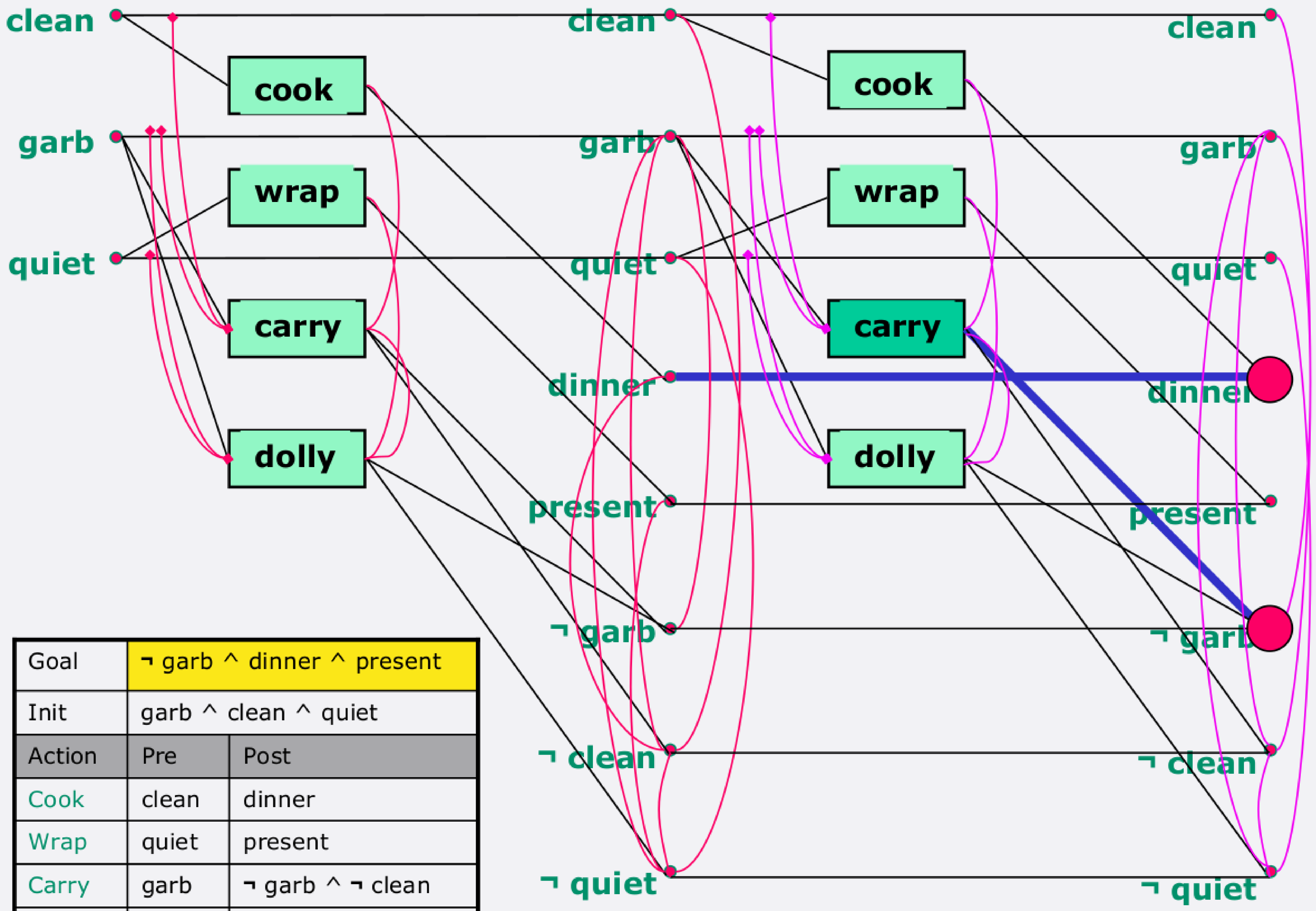
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Starting with not garbage, let's try to satisfy it with carry.

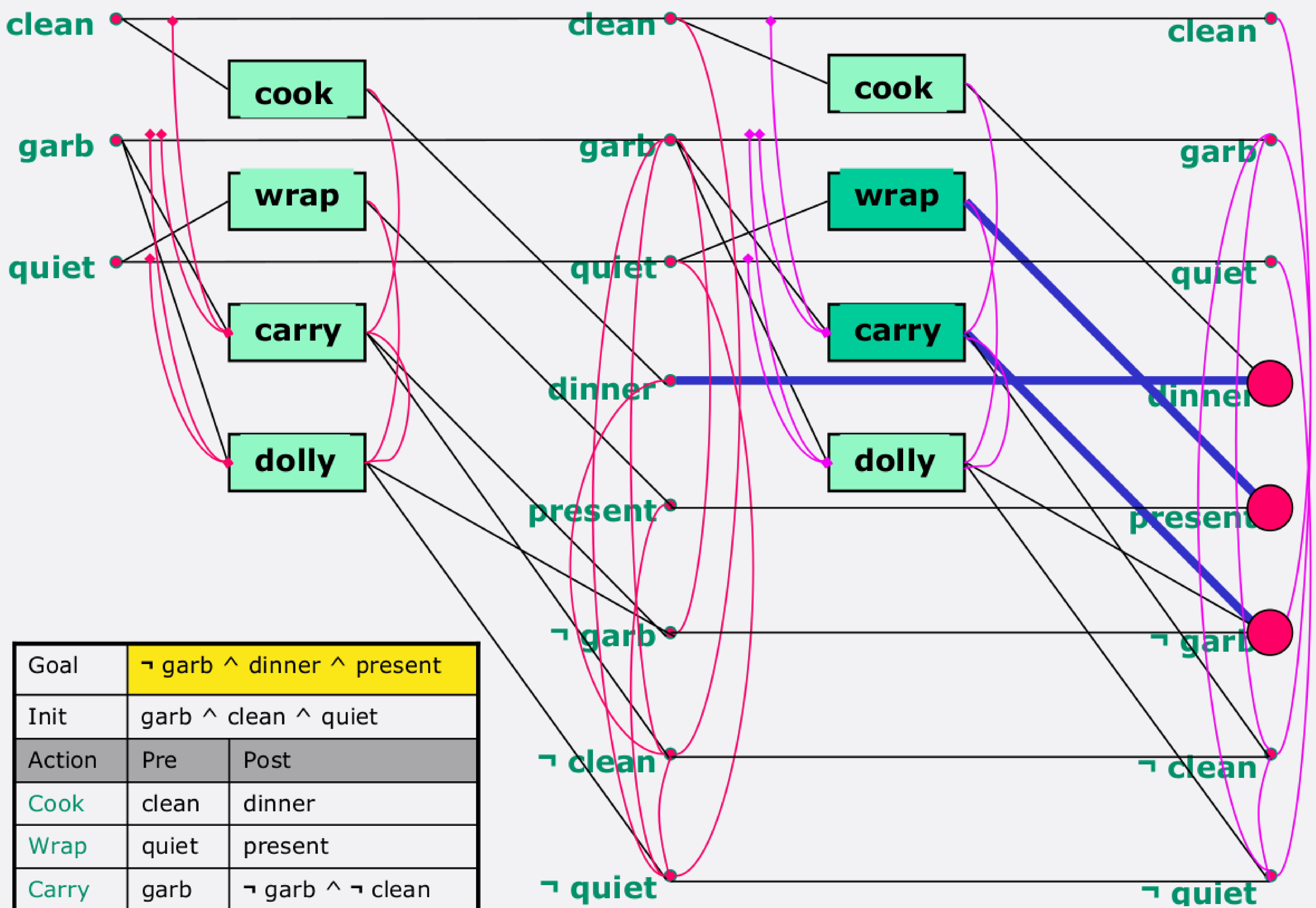
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now we need to satisfy dinner. Since we already know that cook won't be compatible with carry at this level, let's try maintaining dinner from the previous time step. (Of course, it's hard to make a computer as clever as we are, but these are the kinds of tricks that people do when they're making a planner really work efficiently).

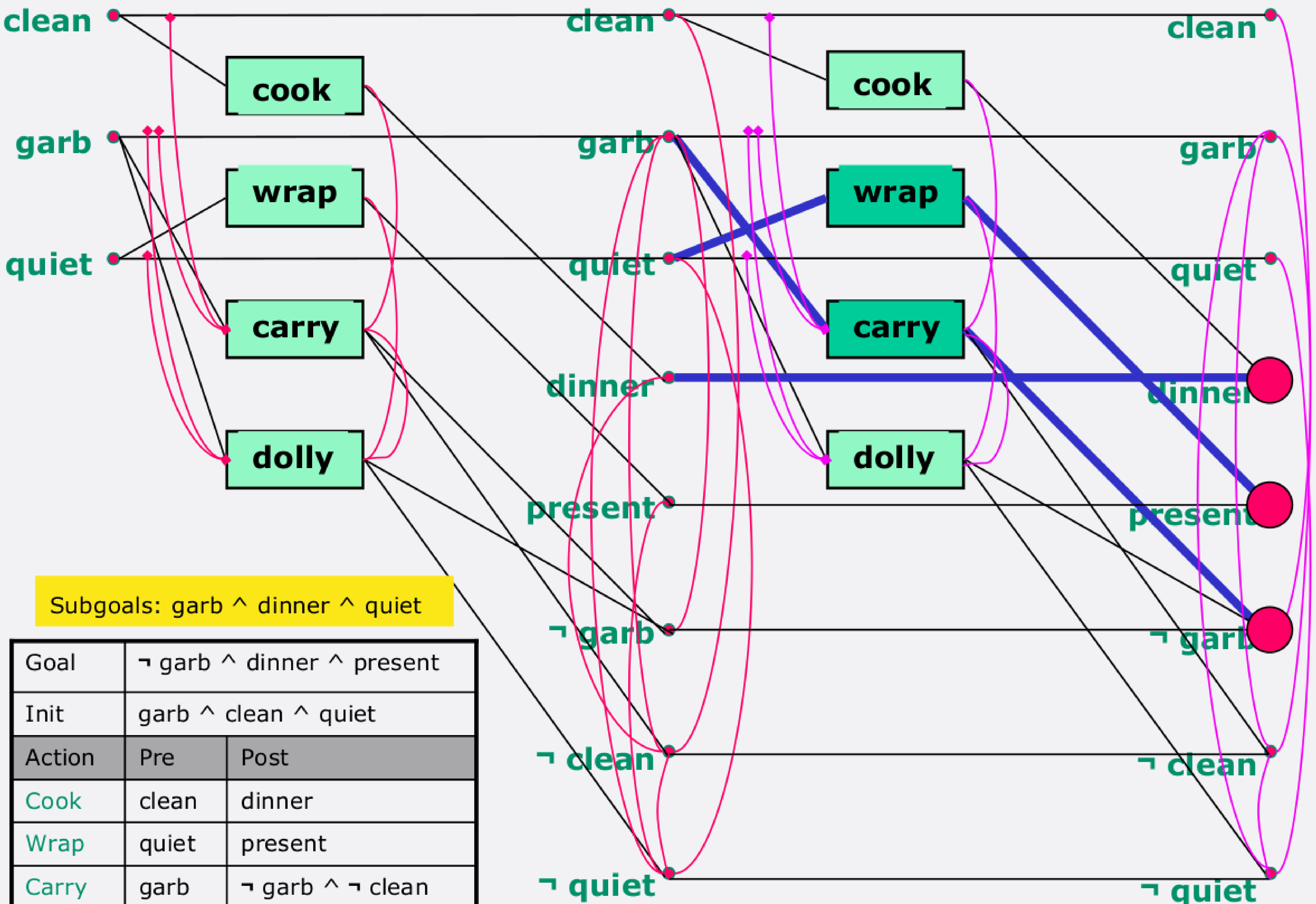
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Last, we need to satisfy present. Let's try doing it with wrap.

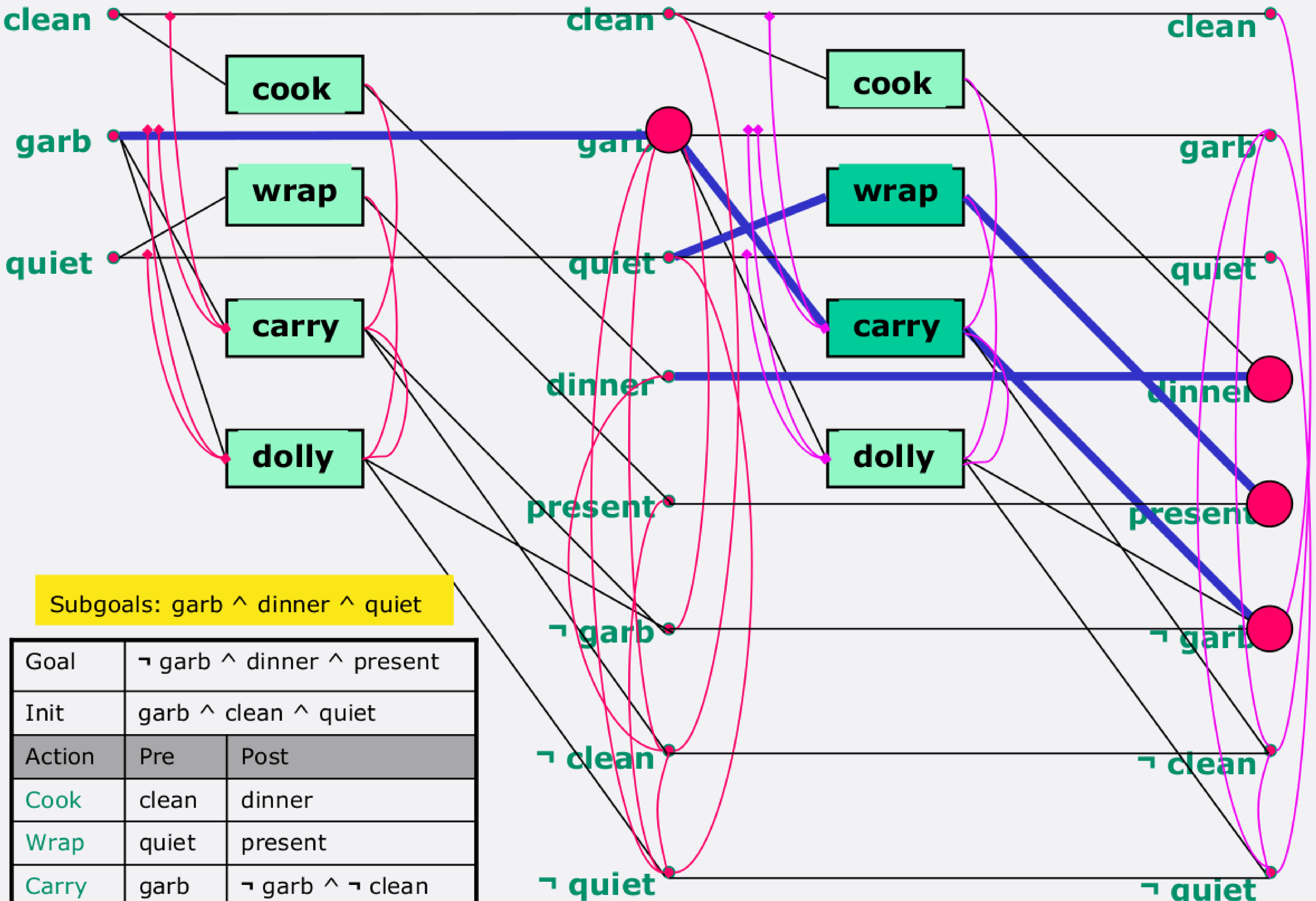
Exercise 10.5



Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We found a way to satisfy all of our conditions at level 4. So now we have to take all the preconditions of the actions we picked and see if we can satisfy them at level 2. Now our subgoals are garbage and dinner and quiet.

Exercise 10.5

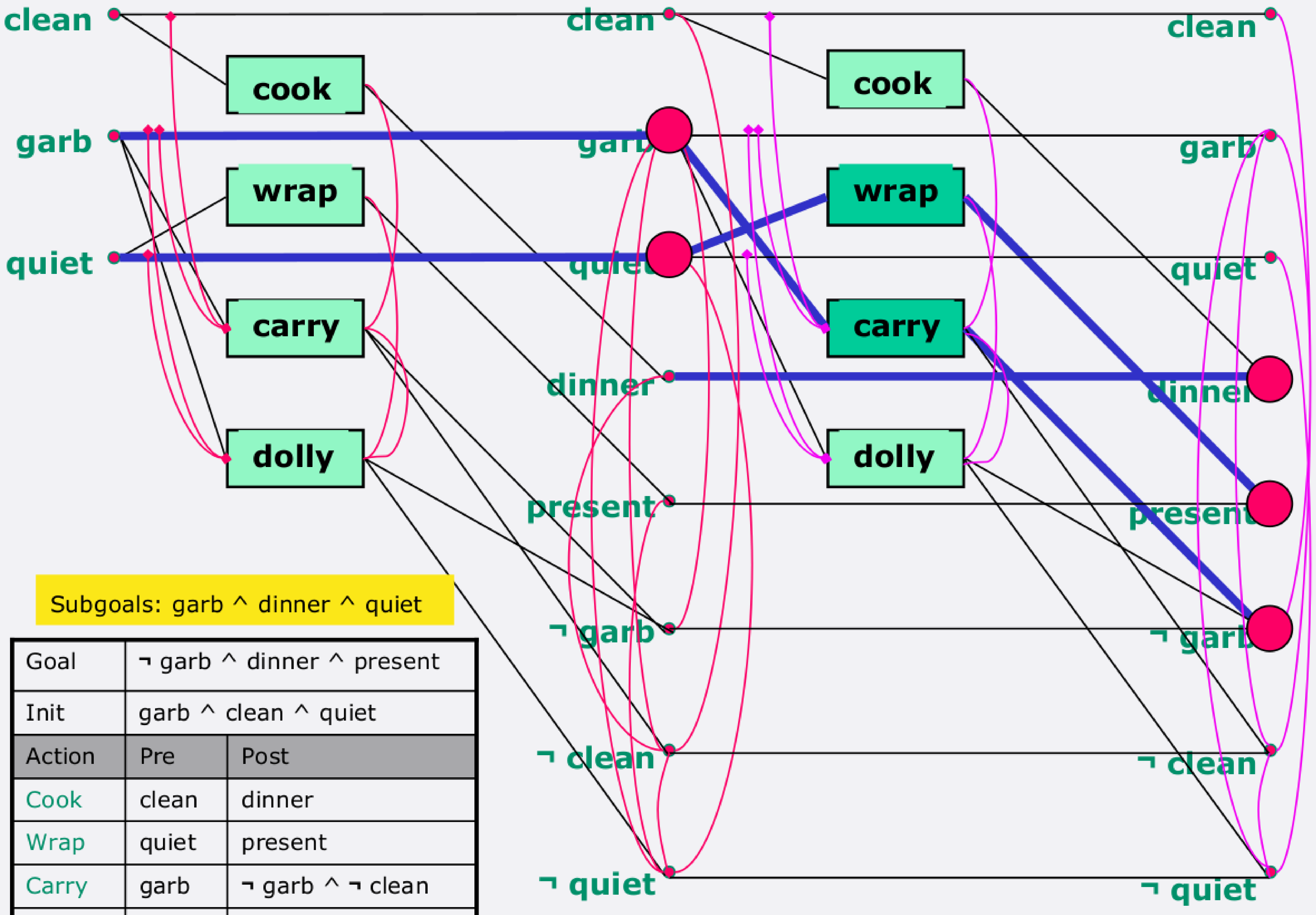


Subgoals: garb ^ dinner ^ quiet

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Let’s start by satisfying garbage by maintaining it. (We don’t have any way to make garbage. Though usually when I cook, it makes garbage).

Exercise 10.5

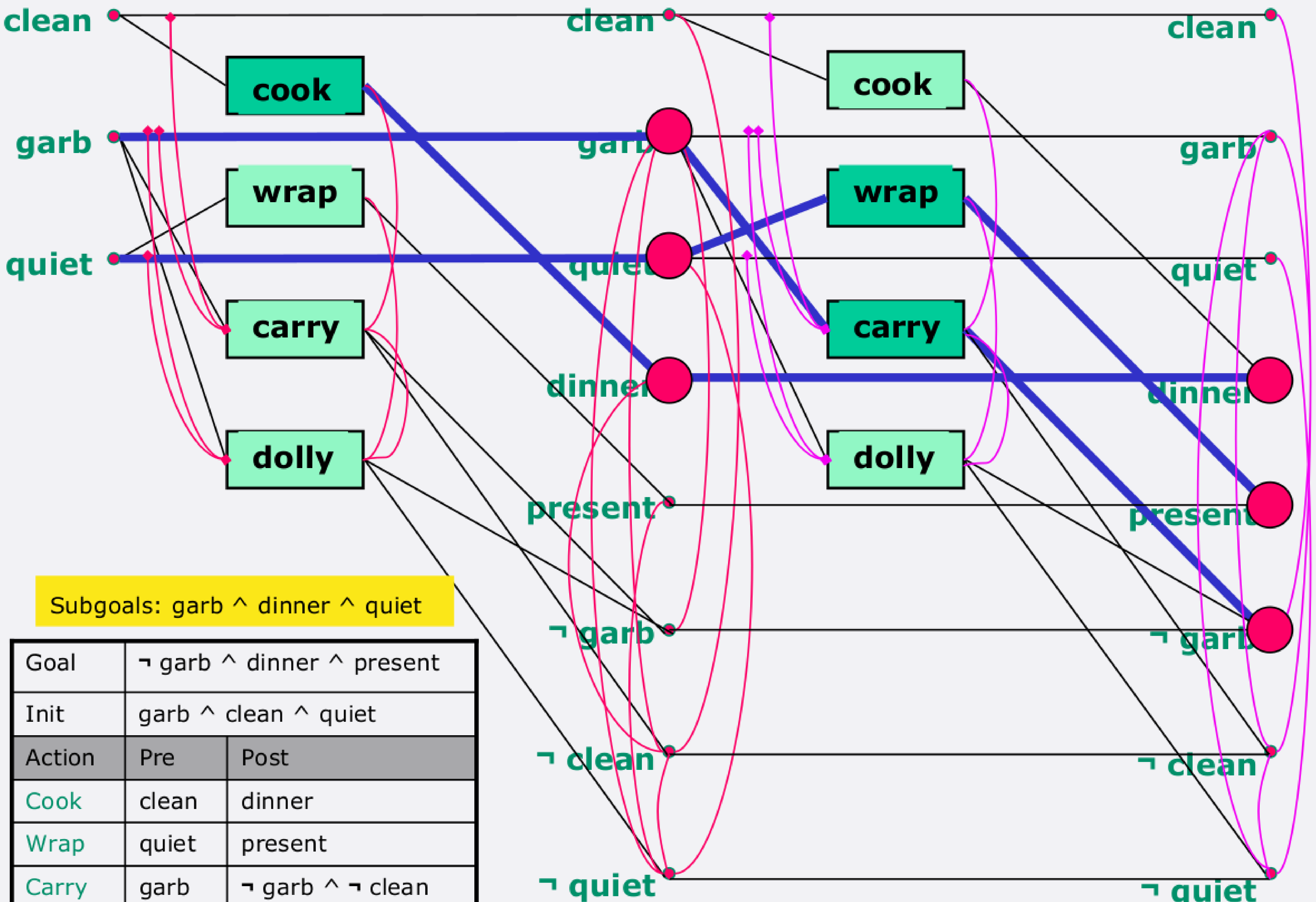


Subgoals: garb ^ dinner ^ quiet

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

We can also easily satisfy quiet by maintaining it.

Exercise 10.5

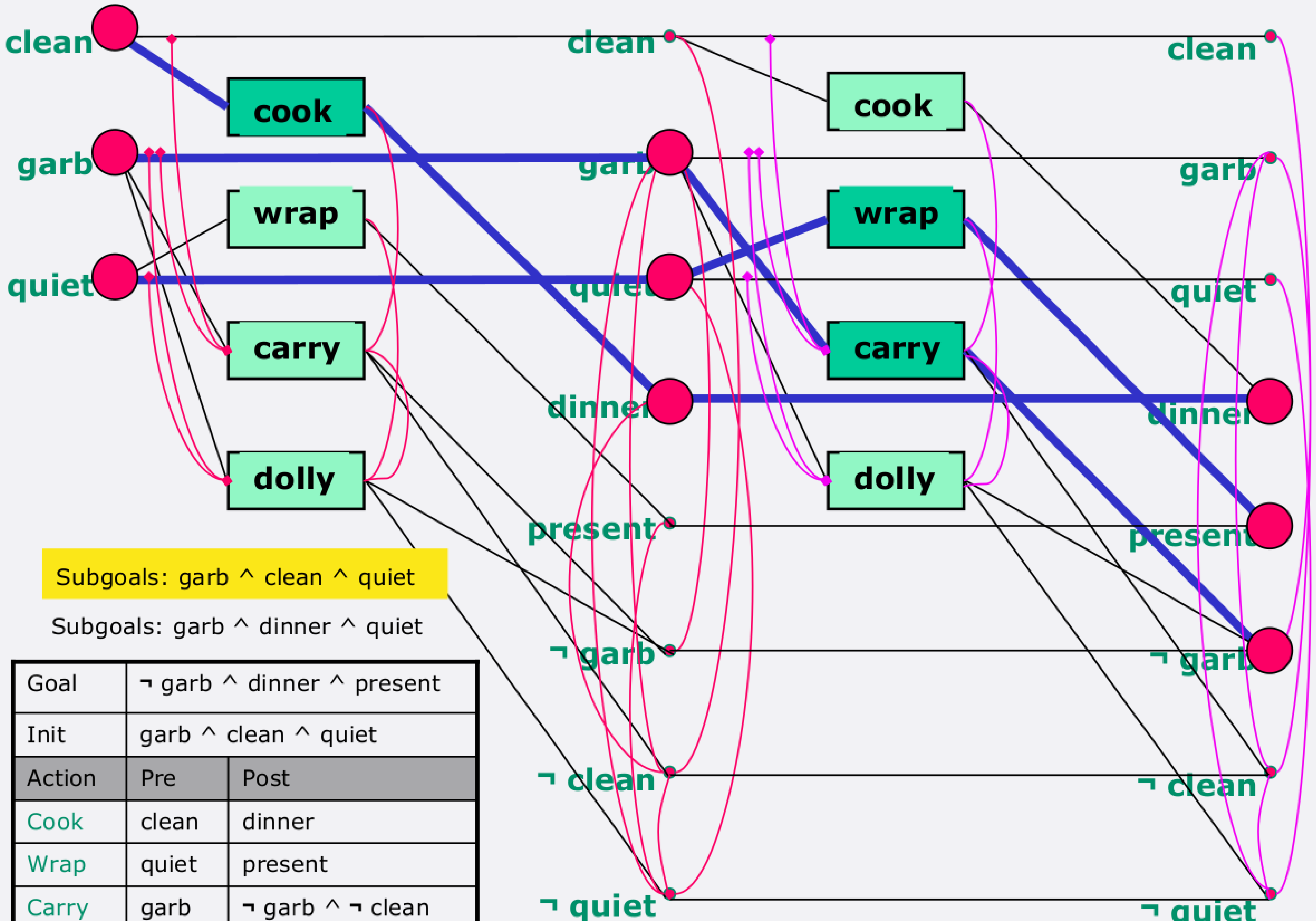


Subgoals: garb ^ dinner ^ quiet

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

And we can satisfy dinner with the cook action.

Exercise 10.5



Subgoals: garb ^ clean ^ quiet

Subgoals: garb ^ dinner ^ quiet

Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Now we have to be sure that we can satisfy all of these preconditions at level 0. Our subgoals now are garbage, clean, and quiet. They're all true at level 0, so we're done! There were actually a lot of plans that would have worked, but here's one of them. If we're going to do actions in order, this plan will allow us to do cook then wrap then carry, or cook then carry, then wrap. The crucial thing is that it forces us to do cook before carry, which we couldn't enforce in a depth 1 plan.