Künstliche Intelligenz

Planung

Dr.-Ing. Stefan Lüdtke

Universität Leipzig

Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI)

Planung

- Planung: Erzeugung einer Sequenz von Aktionen, sodass ein Ziel erreicht wird
- Haben schon Beispiel für Planung gesehen: Suchalgorithmen in Kapitel 3
- Heute: Einführung einer formalen Sprache für Planungsprobleme
- Erlaubt Nutzung von Heuristiken und spezialisierten Algorithmen, sodass wesentlich größere Probleme als mit allgemeinen Suchalgorithmen gelöst werden können

Planung

- Ein **Planungsproblem** ist eine Beschreibung von:
  - dem Ausgangszustand
  - dem Zielzustand
  - einer Menge möglicher Aktionen
- **Planung** ist die Erzeugung einer Sequenz von Aktionen, die vom Start- zum Zielzustand führt
- Eine solche Sequenz bezeichnen wir auch als **Plan**

STRIPS Sprache

- **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver (1972)
- Repräsentation von Zuständen und Zielen:
  - Konjunktion propositionaler und erststufiger funktionsfreier Grundliterale
    z.B. $(Poor \wedge Unknown) \vee Have(Money)$
  - Ziel: Partielle Zustände
  - closed world assumption
- Repräsentation von Operatoren:
  - Name und Parameter
  - Vorbedingung (Konjunktion funktionsfreier Literale)
  - Effekt (Konjunktion funktionsfreier Literale)
  - Operatorschema erlaubt Variablen
  - Beispiel:
    BuyAt(p,x)
    Precondition: *At(p)*, *Sells(p,x)*, *Have(Money)*
    Effect: *Have(x)*, $\neg Have(Money)$

PDDL

- **P**lanning **D**omain **D**efinition **L**anguage (1998)
- Standardisierung von Planungssprachen
- Basiert auf STRIPS

Bestandteile von PDDL

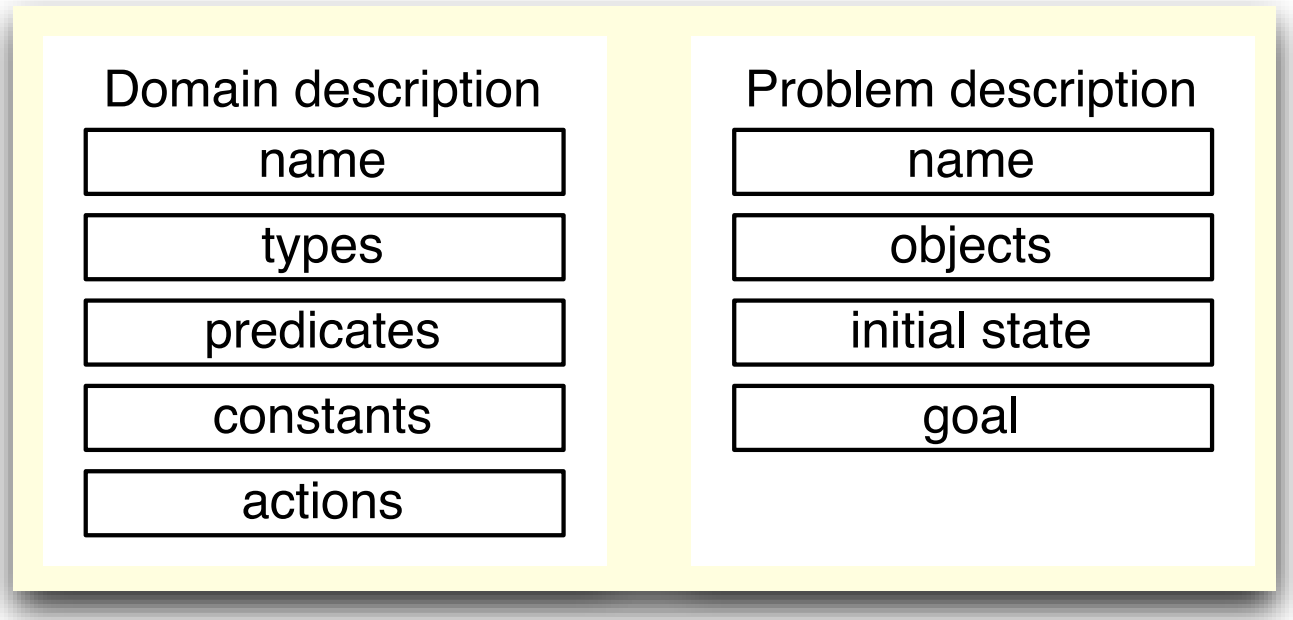
Eine Planungsaufgabe in PDDL besteht aus folgenden Elementen:

- **Objekten:** Dinge in der Welt, die wir modellieren wollen
- **Typen:** Die Klassen (Typen), zu denen die Objekte gehören
- **Prädikate:** Eigenschaften der Objekte, wahr oder falsch
- **Startzustand**
- **Ziel-Spezifikation:** Menge von Prädikaten, die erfüllt sein müssen
- **Aktionen**

PDDL Dateistruktur

Spezifikation ist in zwei Dateien unterteilt:

- **Domain File:** Enthält Typen, Predikate, Aktionen
- **Problem File:** Enthält Objekte, Start- und Zielzustand



Domain File

Das Domain File hat folgende Struktur

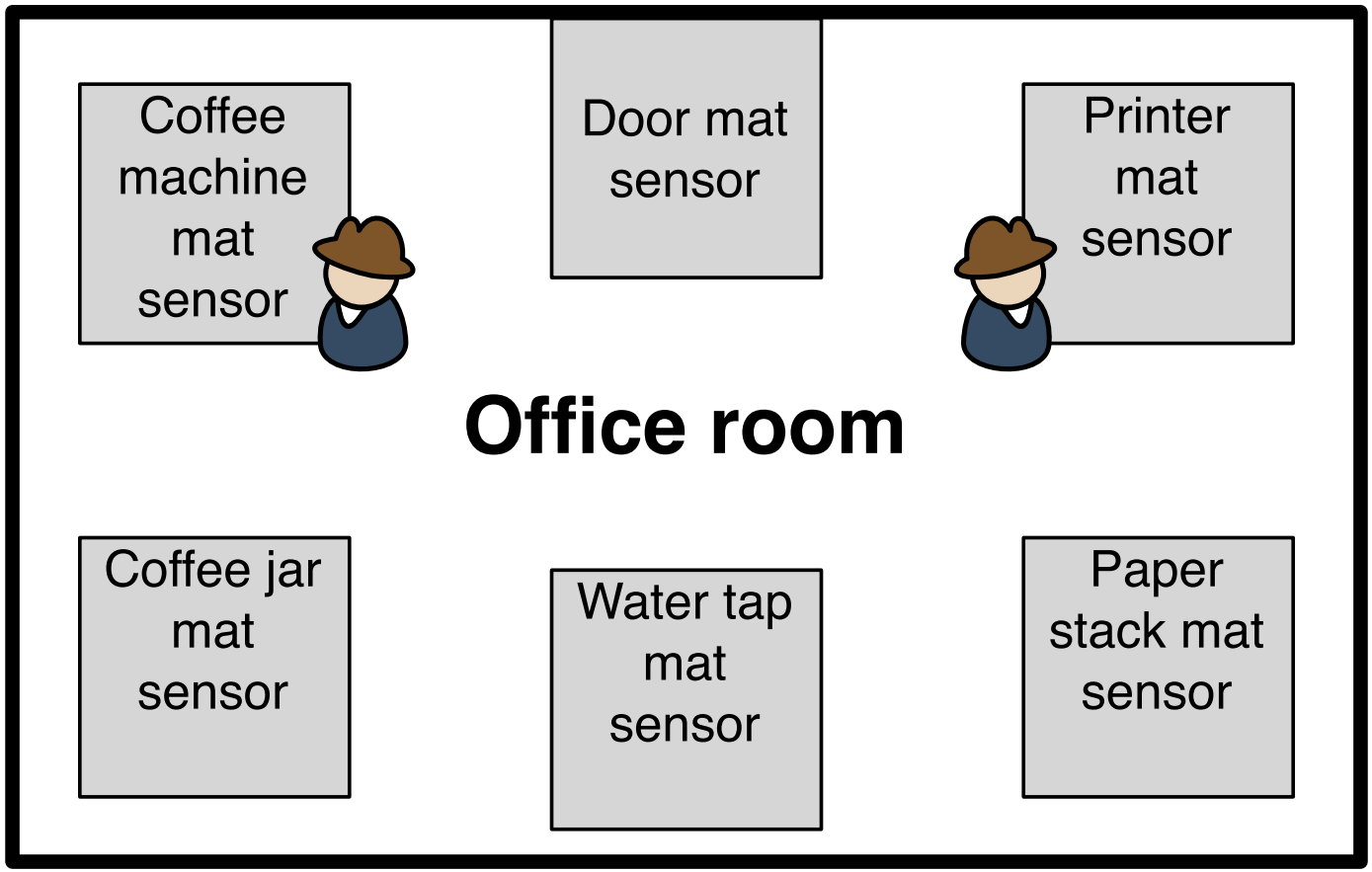
```
domain = (define (domain name){domain-entry})

domain-entry  =
        (:types {name{name} - name}{name})
        | (:predicates  {(name[declarations])})
        | (:constants {constant{constant} - type-name}{
            constant})
        | (:action {action-formula})

type-name      = name
declarations   = {{var} - type-name}{var}
```

Problem File

Das Problem File hat folgende Struktur:

```
problem = (define (problem name){problem-entry})

problem-entry =
      (:domain name)
      | (:objects {constant{constant} - type-name}{
          constant})
      | (:init [:duration atomic-formula]{init-elem})
      | (:goal formula)
```

Beispiel


Coffee machine mat sensor
Door mat sensor
Printer mat sensor
Office room
Coffee jar mat sensor
Water tap mat sensor
Paper stack mat sensor

Beispiel: Domain Description

```
(define (domain office)
  (:types person location object)

  (:predicates
    (has-printed ?p - person ?d - object)
    (at ?p - person ?l - location))

  (:action print
    :parameters (?p - person ?d - object)
    :precondition (and
        (not (has-printed ?p ?d))
        (at ?p printer))
    :effect (and (has-printed ?p ?d))
  )
)
```

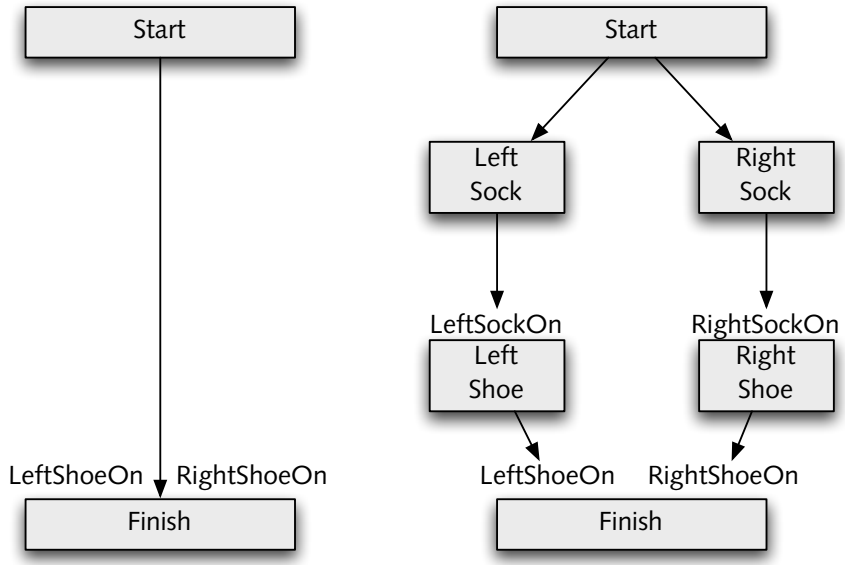
Beispiel Example: Problem Description

```
(define (problem office-1-person)
  (:domain office)

  (:objects
    door printer - location
    john - person
    document - object)

  (:init
    (not (has-printed john document))
    (at john door)
  )

  (:goal (and
    (has-printed john document)
    (at john door))
  )
)
```

Zustandsraum vs. Planungsraum

- Standard Suche: Knoten = Zustand
- Planungssuche: Knoten = Partieller Plan
- Ein partieller Plan besteht aus
  - Eine Menge von Aktionsschemata S_i ("Schritte)
  - Eine partielle zeitliche Anordnung $S_i \prec S_j$
  - Werte für die Variablen in S_i
  - **Kausale Links** $S_i \xrightarrow{c} S_j$ zeichnen auf, dass S_i die Vorbedingung von S_j realisiert
- Operationen mit partiellen Plänen
  - **Hinzufügen eines kausalen Links** von einem existierenden Planungsschritt zu einer offenen Vorbedingung
  - **Hinzufügen eines Schrittes**, um eine offene Vorbedingung zu erfüllen
  - **Anordnen** der Schritte untereinander

Partially Ordered Plans (POP)



- Spezielle Schritte mit leeren Aktionen
  - *start:* Keine Vorbedingung, initialer Zustand als Effekt
  - *finish:* Ziel als Vorbedingung, kein Effekt
- Ein Plan ist **vollständig**, wenn alle Vorbedingungen für jeden Schritt des Plans erfüllt sind
- Eine Vorbedingung c eines Schritts S_j ist erfüllt (über S_i) wenn:
  (i) $S_i \prec S_j$
  (ii) $c \in \mathit{effect}(S_i)$ und
  (iii) Es gibt kein $S_i \prec S_k \prec S_j$ mit $\neg c \in \mathit{effect}(S_k)$
- Ein Plan ist **konsistent**, wenn $S_i \xrightarrow{c} S_j$ eine partielle Ordnung ist

POP Algorithmus

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  $S_{need},\ c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** $S_{need},\ c$

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return $S_{need},\ c$

- Tiefensuche im Raum der partiell geordneten Pläne
- Backtracking über die Alternativen in *Choose-Operator* und *Resolve-Threats*
- **Threat:** Bereits erfüllte Vorbedingung wird durch Linearisierung invalidiert

POP Algorithmus

procedure CHOOSE-OPERATOR($plan$, $operators$, S_{need}, c)

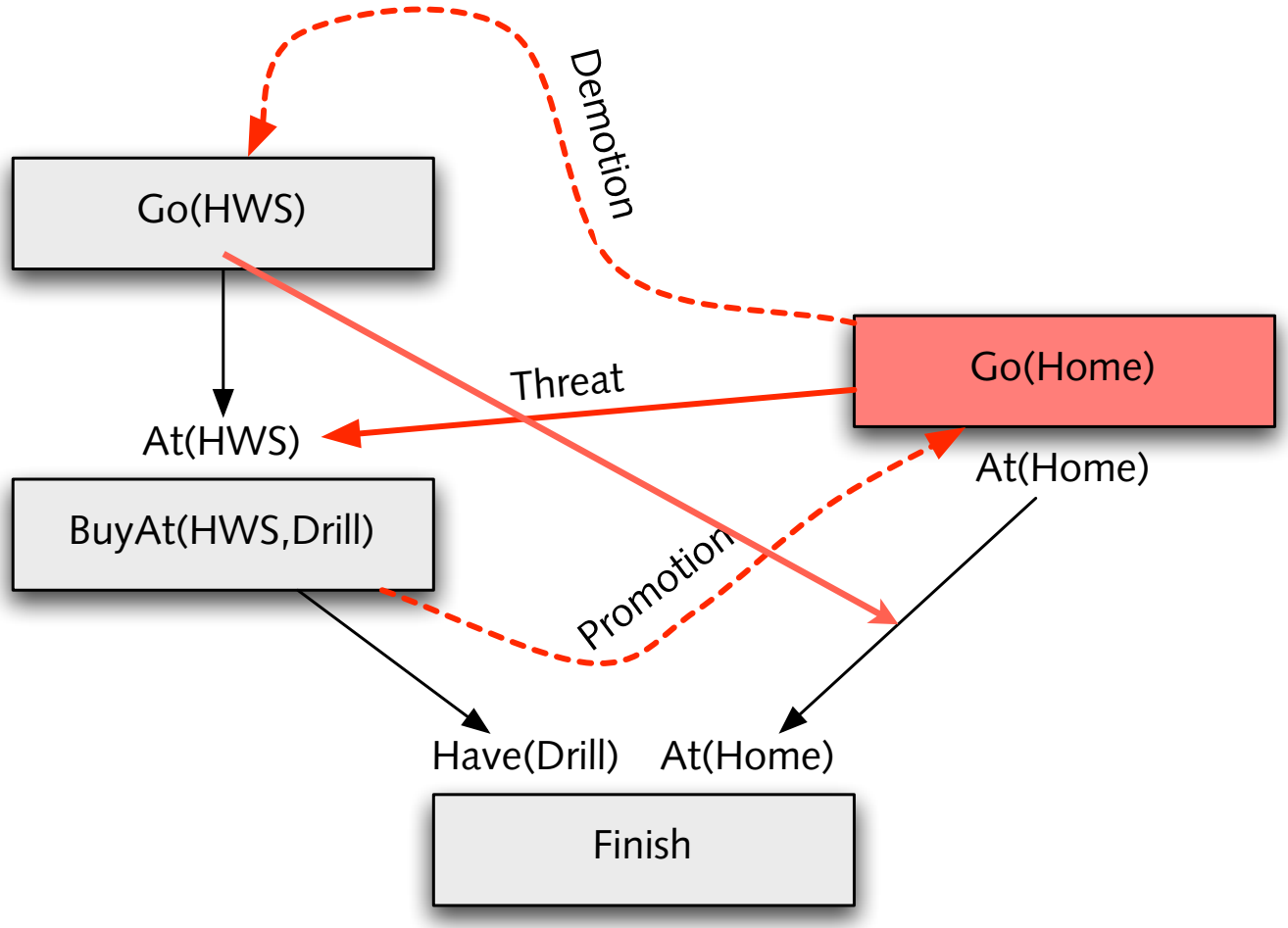
choose a step S_{add} from $operators$ or STEPS($plan$) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
if S_{add} is a newly added step from $operators$ **then**
    add S_{add} to STEPS($plan$)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS($plan$)

---

procedure RESOLVE-THREATS($plan$)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) **do**
    choose either
        Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
        Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
    if not CONSISTENT($plan$) **then fail**
end

Threat

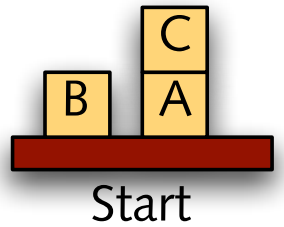


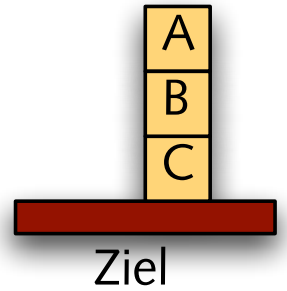
- Ein **Threat** ist ein potentieller Zwischenschritt, der eine durch einen kausalen Link hergestellte Verbindung zerstört
- Lösung: Der Zwischenschritt wird vor oder nach dem kausalen Link angeordnet (promotion oder demotion)

Eigenschaften von POP

- DFS, Backtracking über
  - Alternativen für S_{add}
  - Alternativen für promotion / demotion
  - Alle $c \in \mathit{precond}(S_{need})$ müssen erfüllt werden (keine Auswahl)
- POP ist konsistent, vollständig, und systematisch (keine Wiederholung von Plänen)
- Nützlich für lose Probleme mit lose verknüpften Teilzielen
- Erzeugt partiell geordnete Pläne (warum ist das eine gute Sache?)

POP Beispiel





Clear(x) On(x,z) Clear(y)

PutOn(x,y)

Clear(z) On(x,y)
¬On(x,z) ¬Clear(y)

Clear(x) On(x,z)

PutOnTable(x,y)

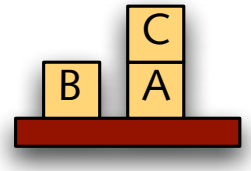
Clear(z) On(x,Table)
¬On(x,z)

- Drei Blöcke A, B, C auf dem Tisch
- agent soll Blöcke so stapel, dass A auf B auf C ist
- Agent kann nur jeweils einen Block bewegen

POP Beispiel

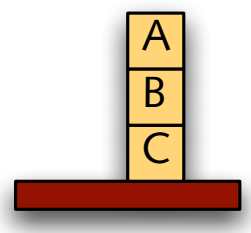
Start

On(C,A) On(A,Table) Clear(B) On(B,Table) Clear(C)

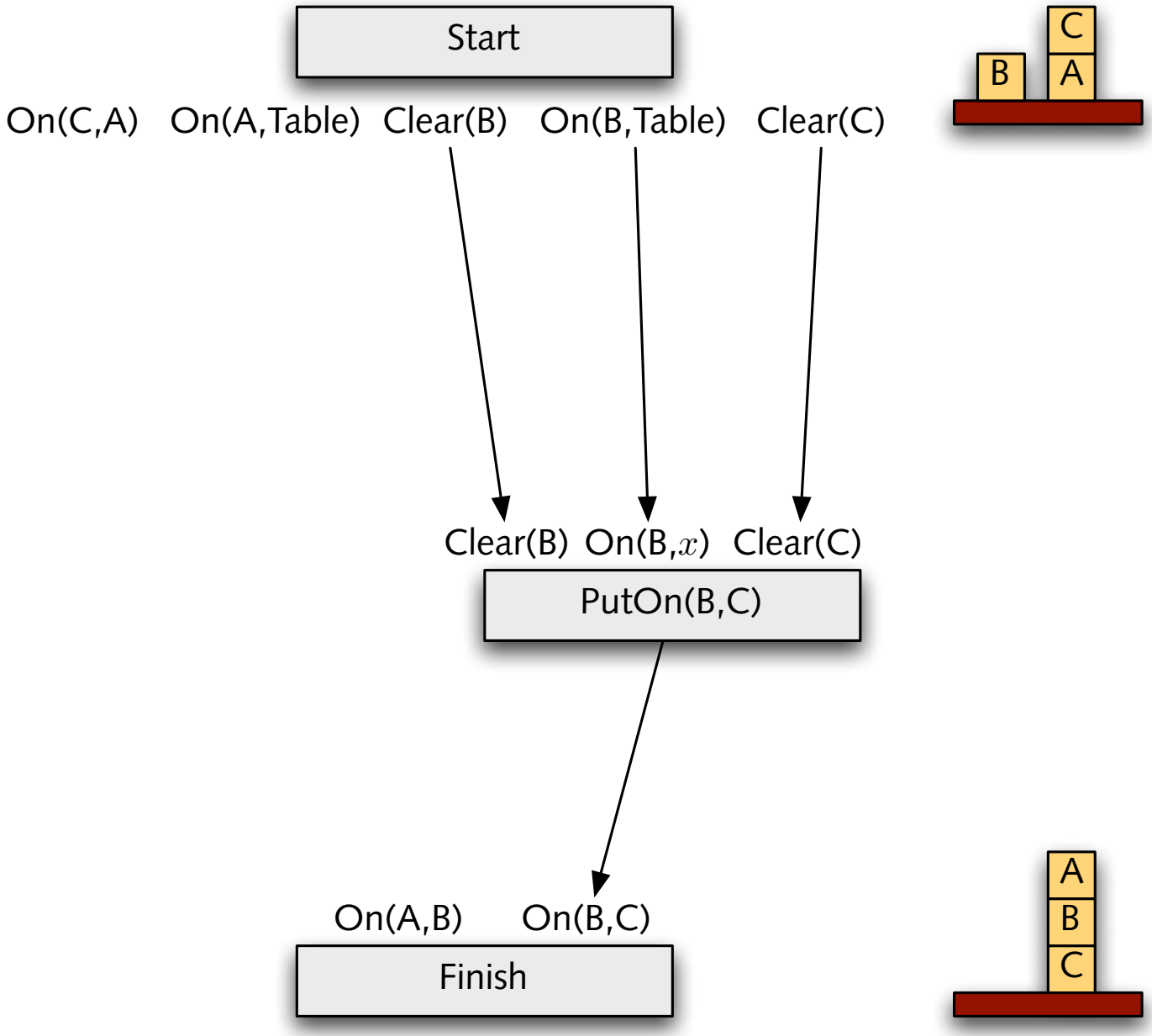


On(A,B) On(B,C)

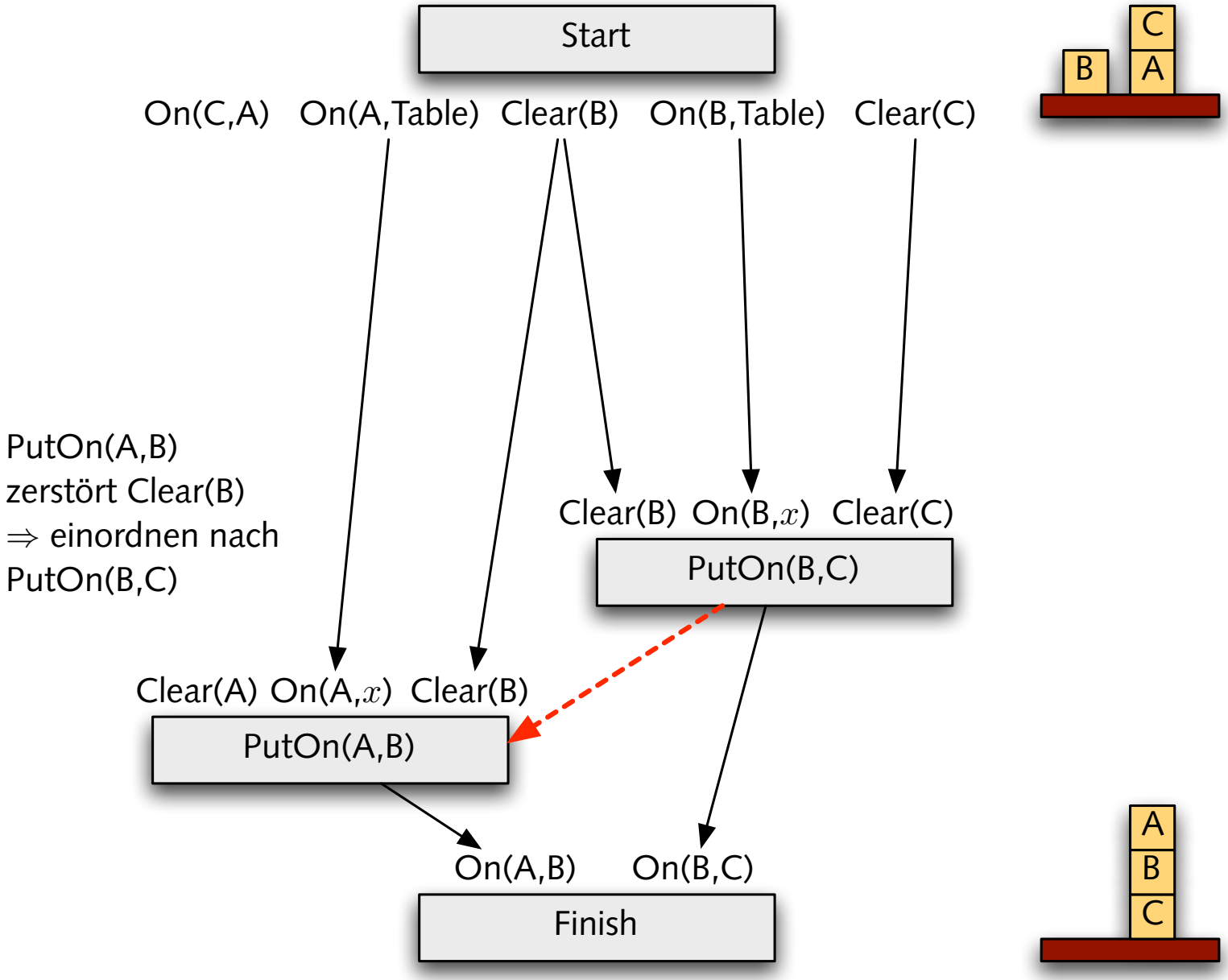
Finish



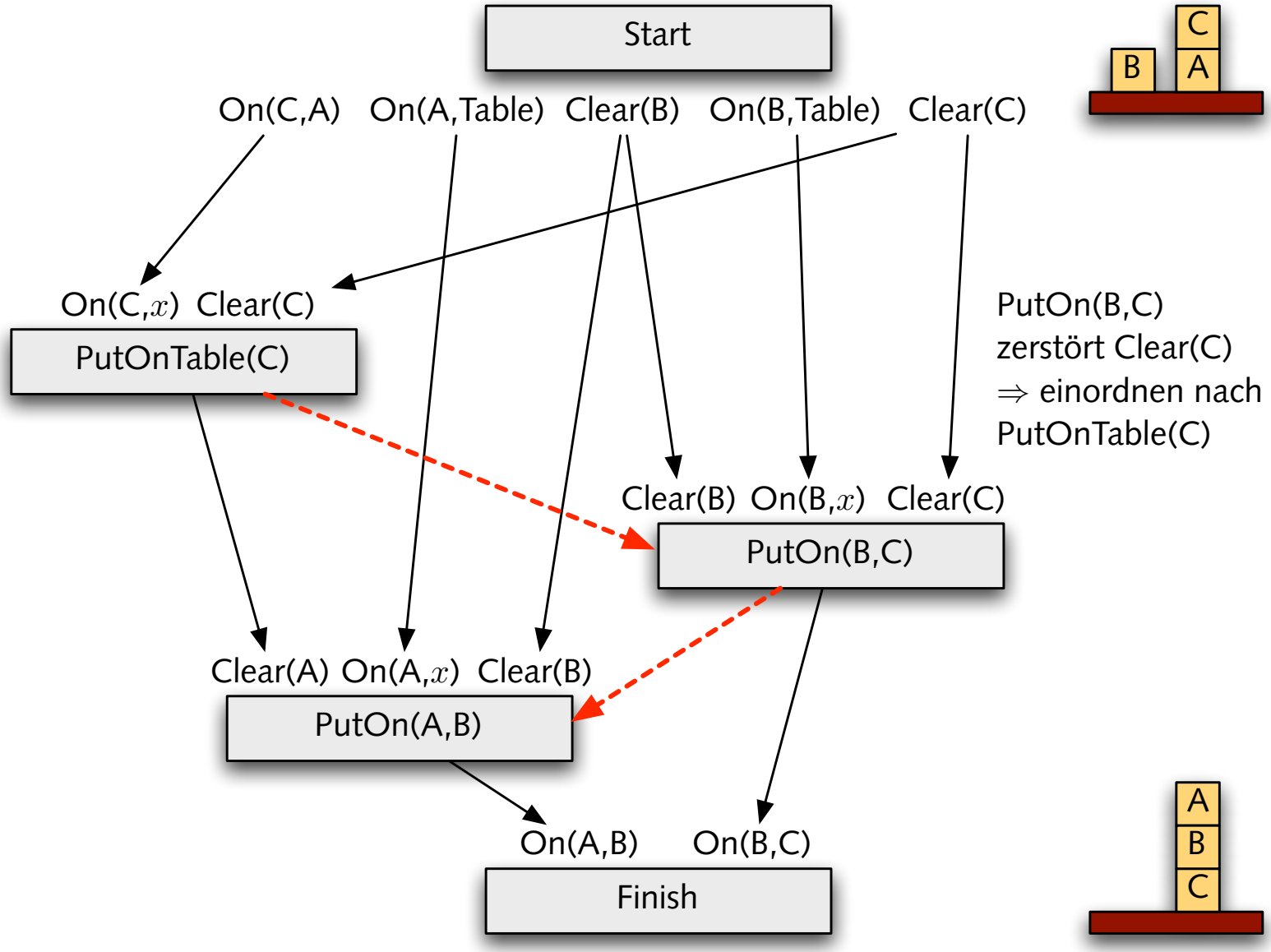
POP Beispiel



POP Beispiel



POP Beispiel



Graph Planning

- POP
  - “menschliche”, natürliche Herangehensweise, aber
  - Komplexe Struktur des Zustandsraums
  - dadurch langsam
- Graph Planning:
  - Propositionaler Planer (keine Variablen)
  - Dadurch einfacher
  - Aber: größerer Suchraum

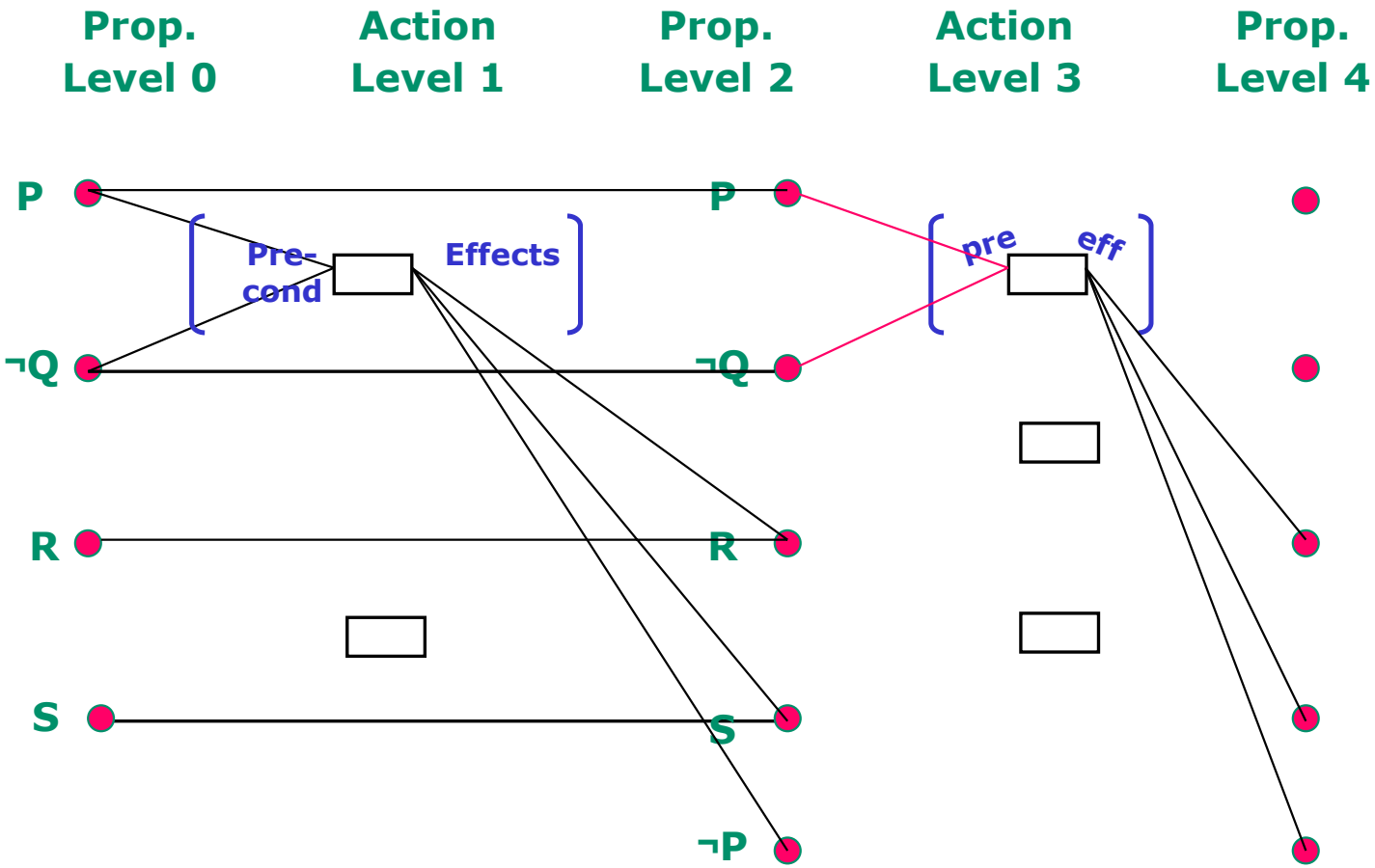
Graph Plan Algorithm

1: **function** GRAPH-PLAN($P, A, g, init, k$)
2: **returns:** a sequence of actions from the initial to the goal state
3: **inputs:** P, a list of propositions
4: A, a list of actions
5: g, goal specification
6: $init$, initial state
7: k, graph depth
8:
9: $graph \leftarrow$ MAKE-PLAN-GRAPH($P, A, g, init, k$) ▷ make a plan graph of depth k
10: $sol \leftarrow$ SEARCH-SOL($graph$) ▷ search for a solution in the graph
11: **if** sol not empty **then**
12: **return** sol
13: $k \leftarrow k + 1$
14: GRAPH-PLAN($P, A, g, init, k$) ▷ call the algorithm with graph depth of $k + 1$

Graph Plan

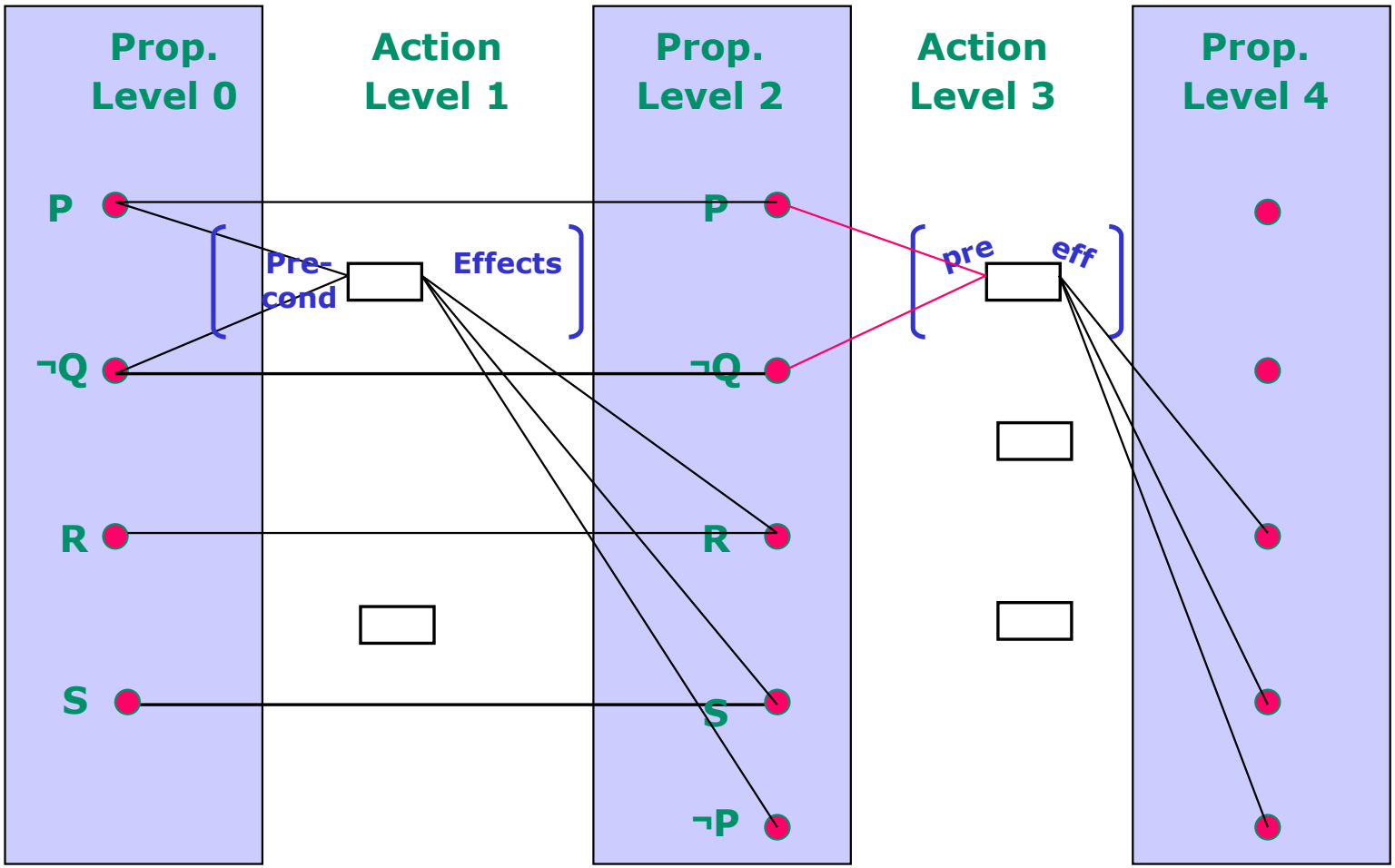
1. Erzeuge Plan-Graphen der Tiefe k
2. Suche nach Lösung
3. Wenn erfolgreich, gib Lösung (=Plan) zurück
4. Ansonsten, $k = k + 1$ und gehe zu Schritt 1

Plan Graph



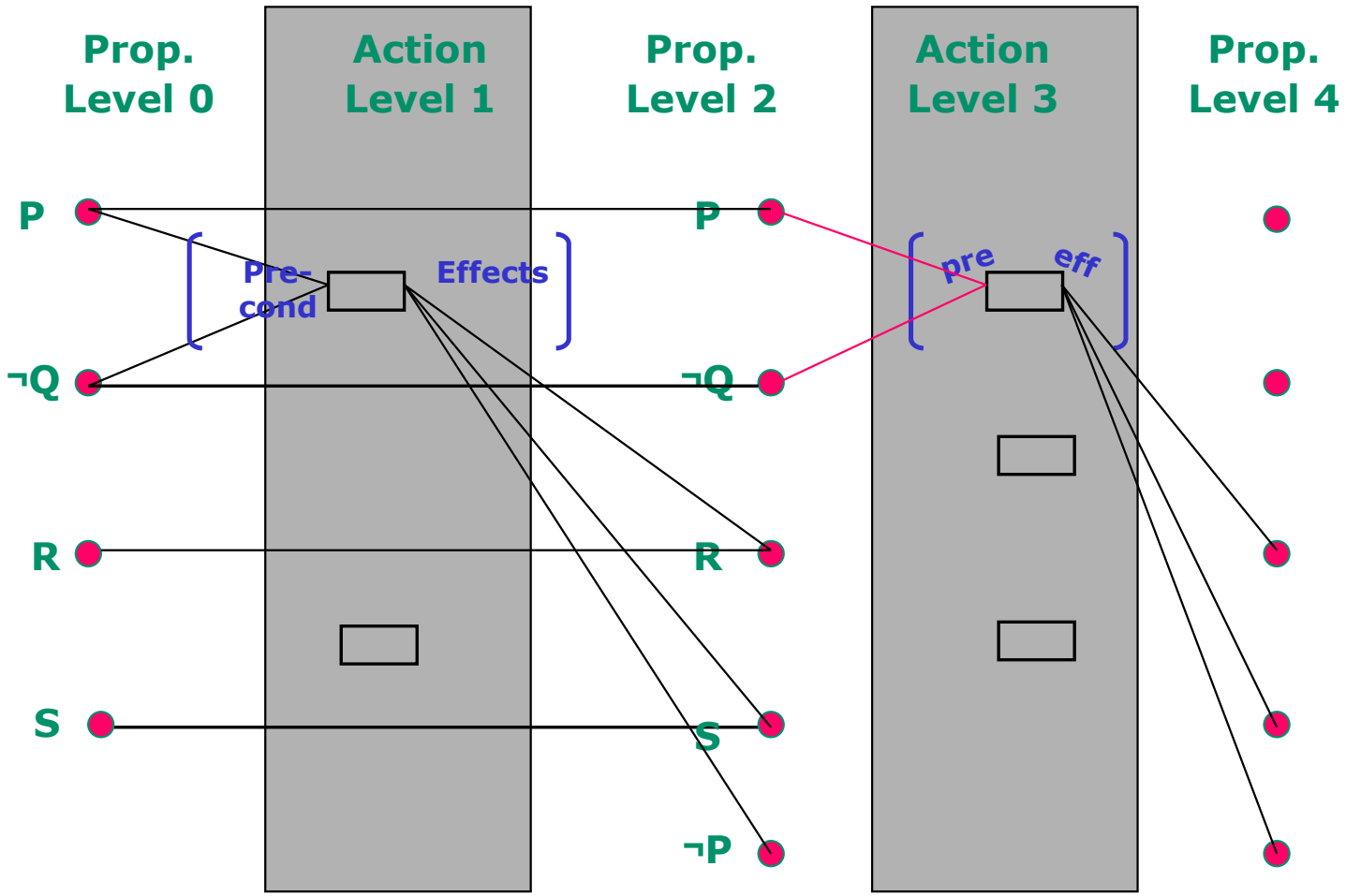
- we search in a plan graph starting with level 0 (the initial state) and increasing the levels until a solution is found

Plan Graph



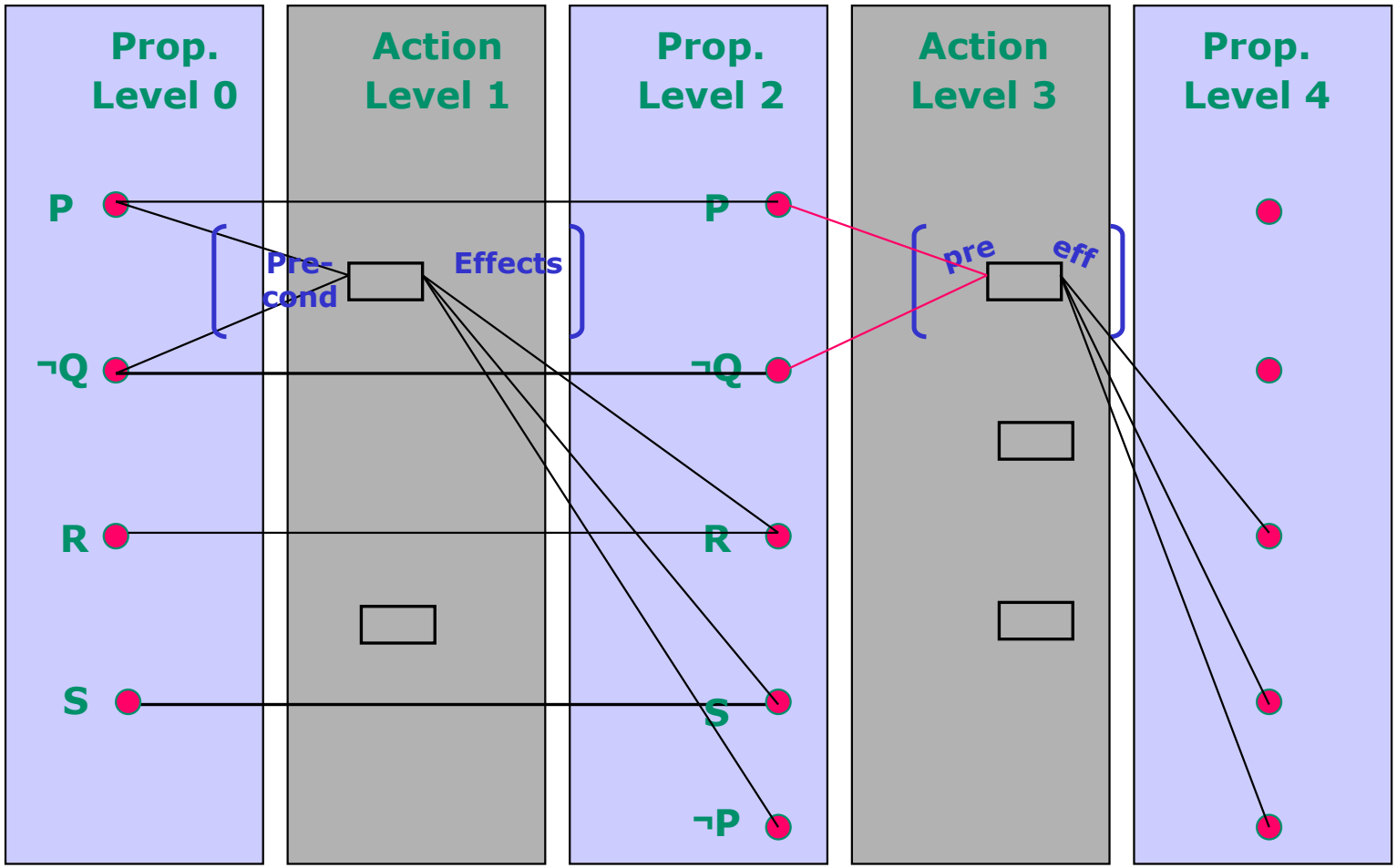
- the even levels contain propositions that describe the state of the world

Plan Graph



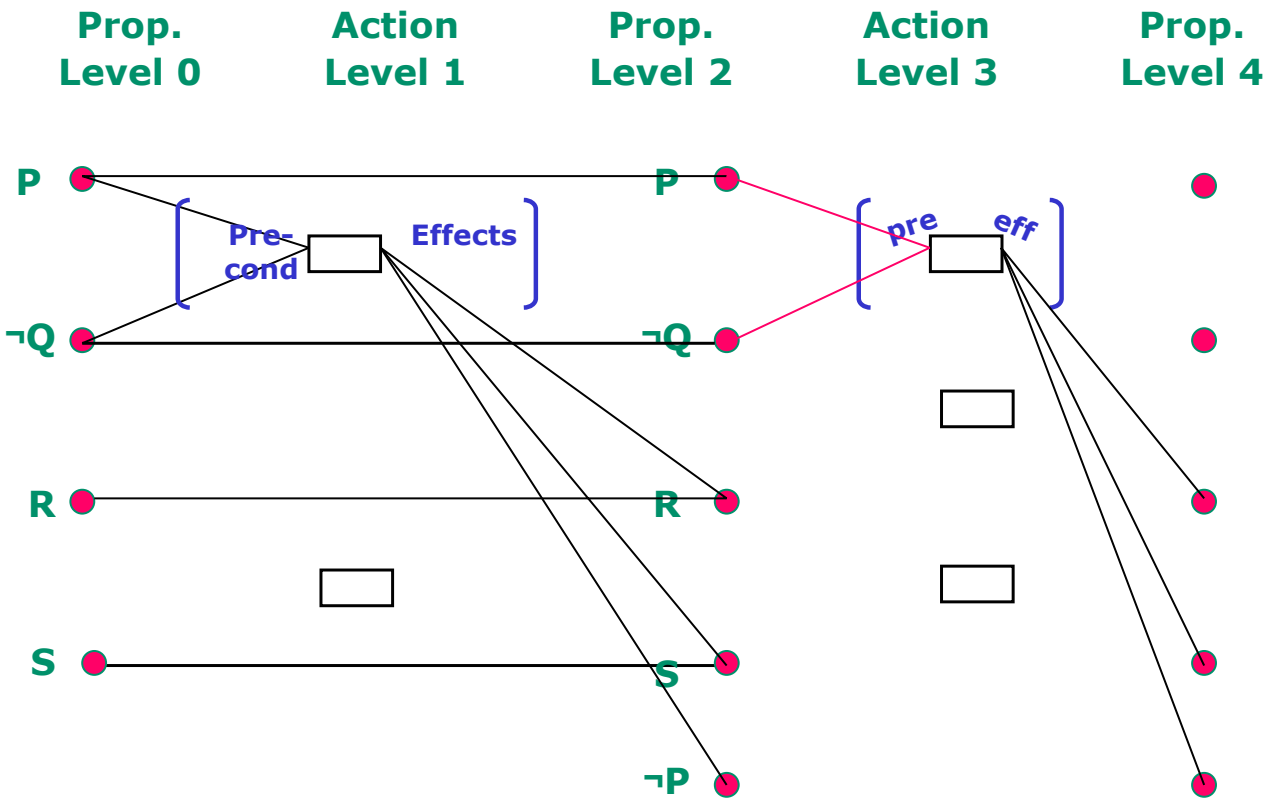
- the odd levels contain actions that change the state of the world

Plan Graph



- in this example we have three propositional and two action levels
- thus we are able to encode level 2 plans as there are only 2 layers with actions

Plan Graph

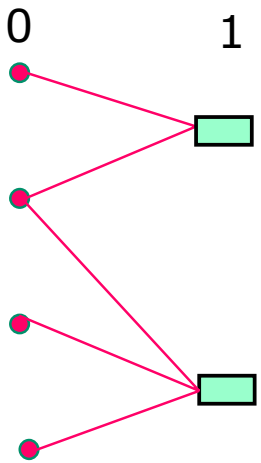


- we start by generating a graph with levels 0 to 2 (i.e. initial state, possible actions, next state)
- this corresponds to plan of depth 1
- in case no solution was found, we extend the graph to level 4 (a plan with depth 2)

Making the Plan Graph

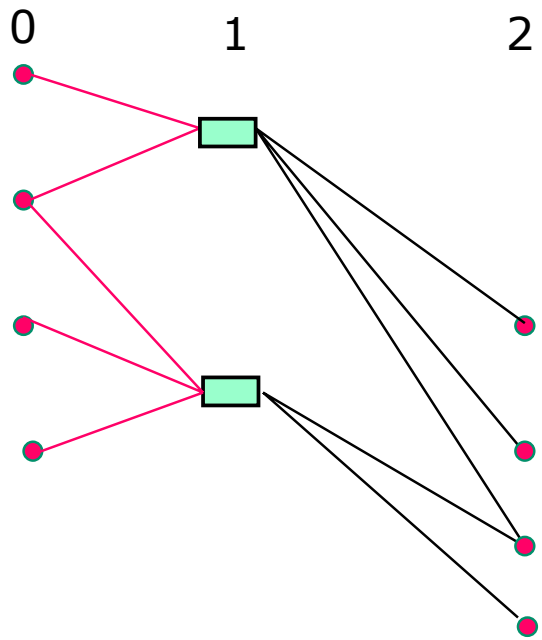
- we start with the initial conditions

Making the Plan Graph



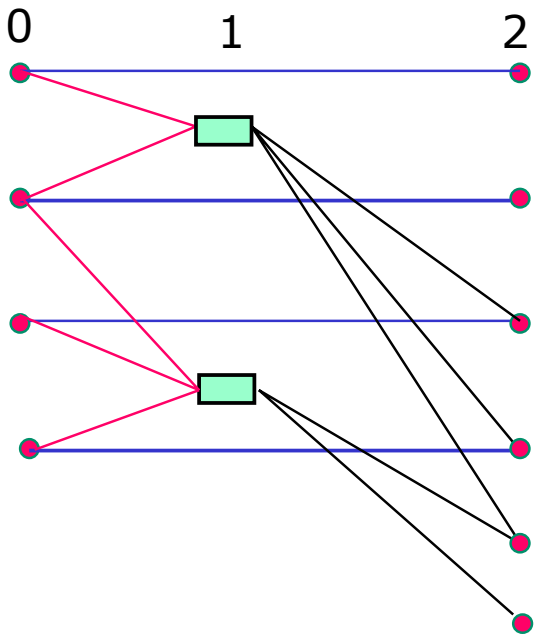
- we start with the initial conditions
- add actions with satisfied conditions

Making the Plan Graph



- we start with the initial conditions
- add actions with satisfied conditions
- add all effects of the actions from the previous level

Making the Plan Graph

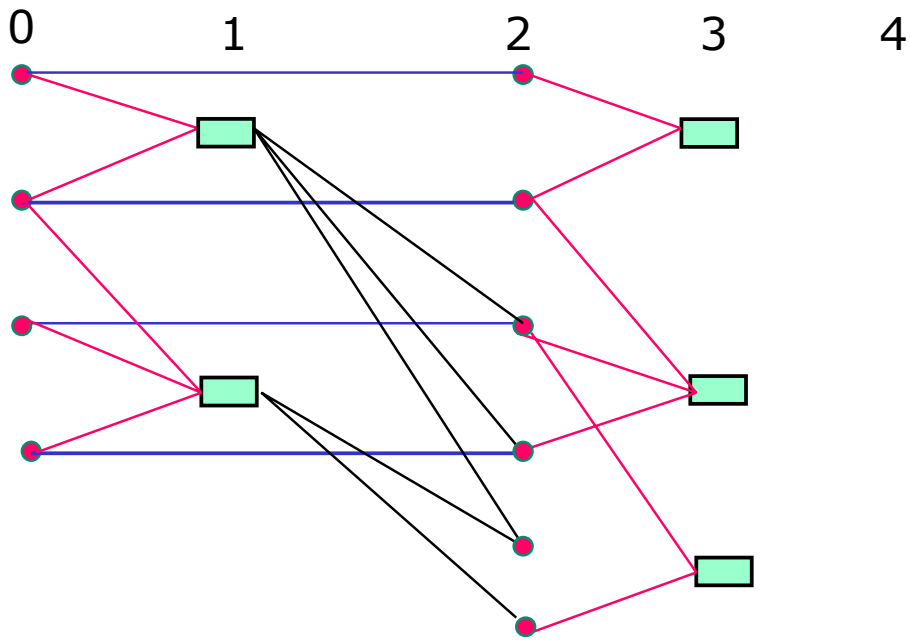


- we start with the initial conditions
- add actions with satisfied conditions
- add all effects of the actions from the previous level
- add all propositions from level 0 to level 2 and add maintenance actions (with blue lines)

Maintenance Actions

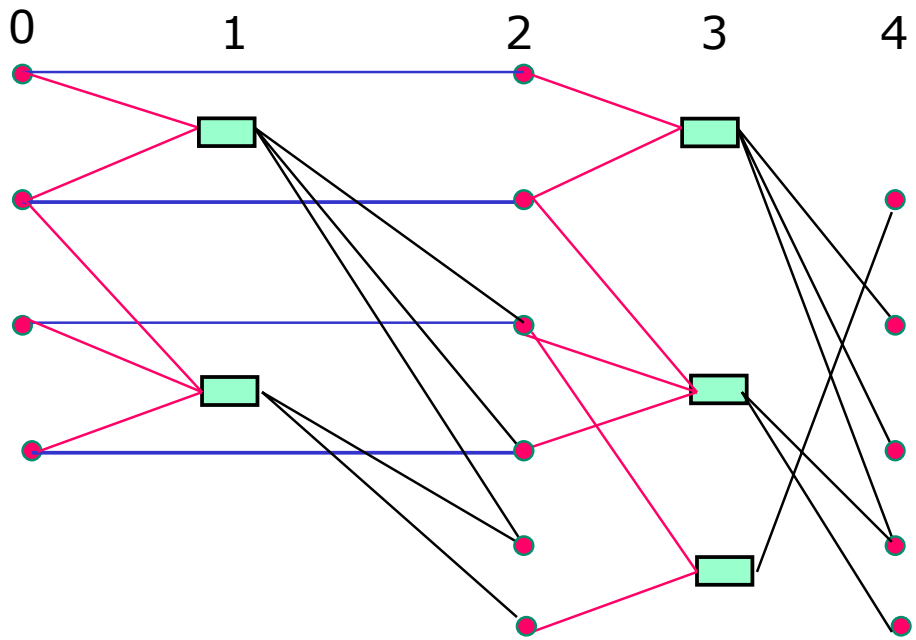
- maintenance actions are actions which transport unchanged propositions from one level to another
- represent the possibility of having some proposition be true at step n because it was true at step $n-2$ and there were no actions to change it
- i.e. they maintain the truth value of unchanged propositions

Making the Plan Graph



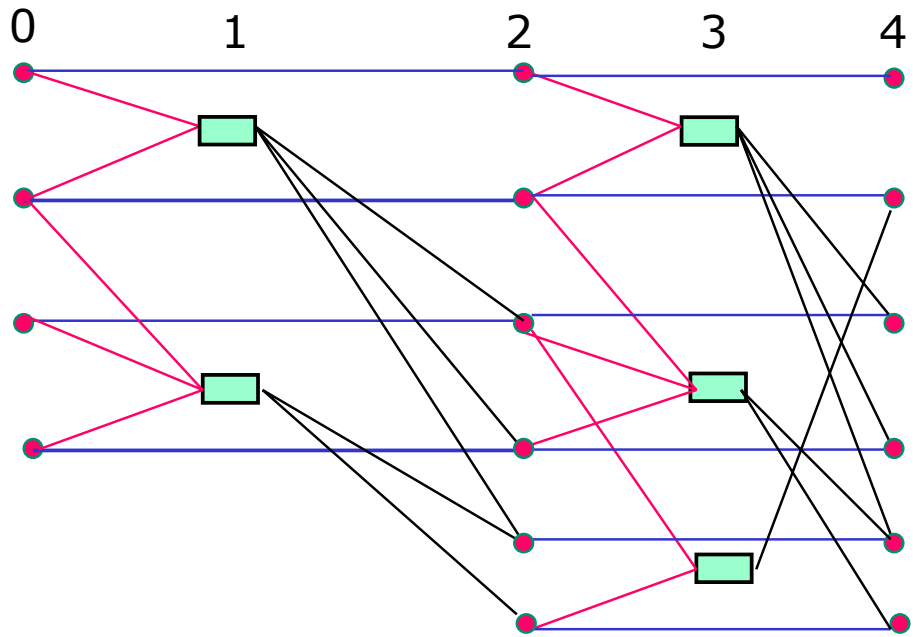
- we start with the initial conditions
- add actions with satisfied conditions
- add all effects of the actions from the previous level
- add all propositions from level 0 to level 2 and add maintenance actions (with blue lines)
- repeat for the next level

Making the Plan Graph



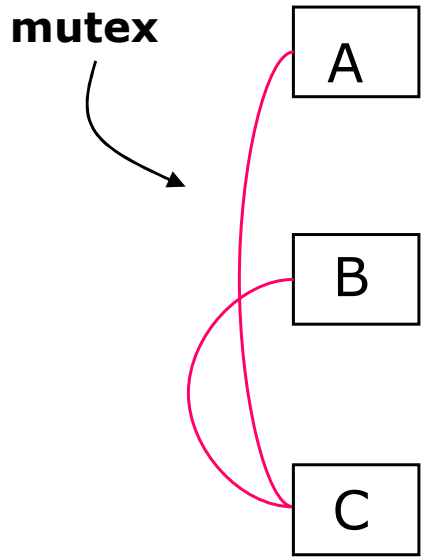
- we start with the initial conditions
- add actions with satisfied conditions
- add all effects of the actions from the previous level
- add all propositions from level 0 to level 2 and add maintenance actions (with blue lines)
- repeat for the next level

Making the Plan Graph



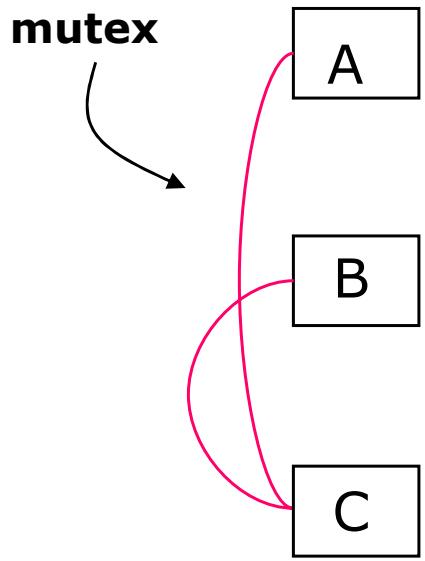
- we start with the initial conditions
- add actions with satisfied conditions
- add all effects of the actions from the previous level
- add all propositions from level 0 to level 2 and add maintenance actions (with blue lines)
- repeat for the next level

Mutually Exclusive Actions



- at this point the graph is representation of the complete search tree down to depth n
- what about mutually exclusive actions?
  - a pruning phase where we find and mark mutually exclusive actions
  - these are actions that cannot be done in the same step
  - we call these actions **mutex**
  - actions A and C are mutex and actions B and C are mutex
  - we could execute A and B in parallel but if we execute C we can't do any of the others

Mutually Exclusive Actions



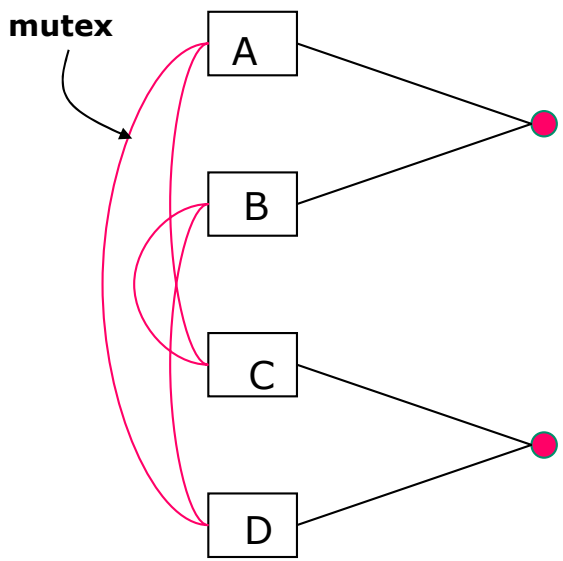
Two actions at level i are mutex in case of:

- **inconsistent effects:** effect of one action is negation of effect of another
- **interference:** one action deletes the precondition of another
- **competing needs:** the actions have preconditions that are mutex at level $i - 1$

Mutually Exclusive Propositions

Two propositions at level i are mutex in case of:

- **negation:** they are negations of one another
  - e.g. P and $\neg P$
- **inconsistent support:** all ways of achieving the propositions at level $i - 1$ are pairwise mutex

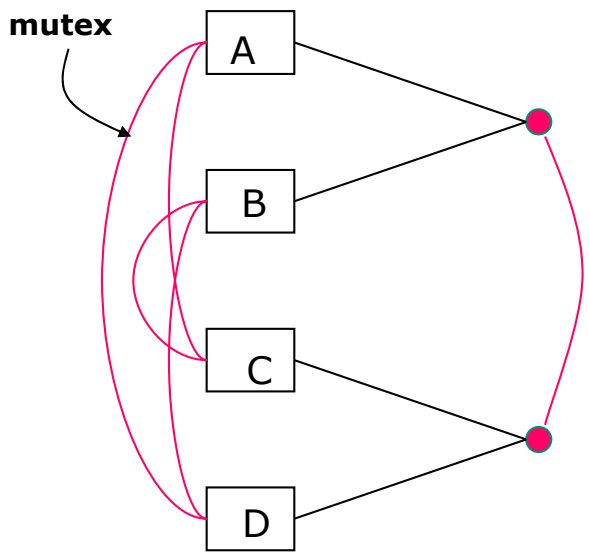


Inconsistent support

Mutually Exclusive Propositions

Two propositions at level i are mutex in case of:

- **negation:** they are negations of one another
  - e.g. P and $\neg P$
- **inconsistent support:** all ways of achieving the propositions at level $i-1$ are pairwise mutex
- the arc indicates that they are mutex



Inconsistent support

Solution Extraction

To find a solution we have the following steps:

- if all the literals in the goal appear at the deepest level and **no** mutex, then we search for a solution for each subgoal at level i
  - e.g. if our goal is P and $\neg Q$, then both P and $\neg Q$ are in the last proposition level; if they are not mutex, we look for a plan
  - for each subgoal at level i
    - choose an action to achieve it
    - if it is mutex with another action, return *fail*
  - repeat for preconditions at level $i - 2$ (our next set of subgoals)

Birthday Dinner example

Goal: get a birthday dinner for somebody who is at home and asleep.

- Goal: $\neg$ garbage and dinner and present
- Init: garbage and clean and quiet
- Actions:
  - Cook
    Pre: clean
    Eff: dinner
  - Wrap
    Pre: quiet
    Eff: present
  - Carry
    Pre: garbage
    Eff: $\neg$ garbage and $\neg$ clean
  - Dolly
    Pre: garbage
    Eff: $\neg$ garbage and $\neg$ quiet

Birthday Dinner example

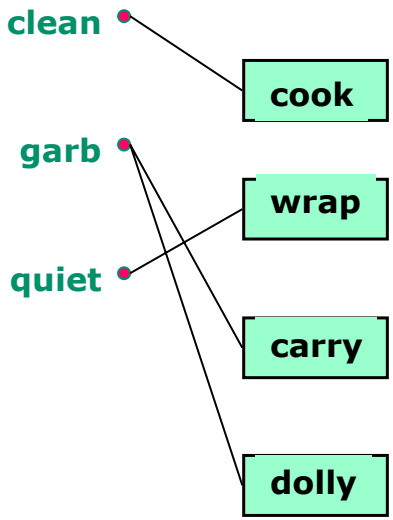
clean

garb

quiet

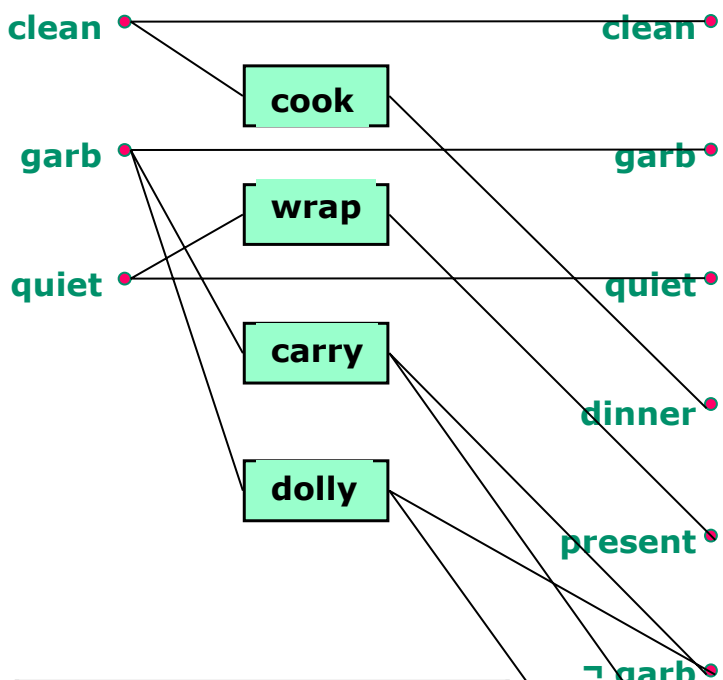
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



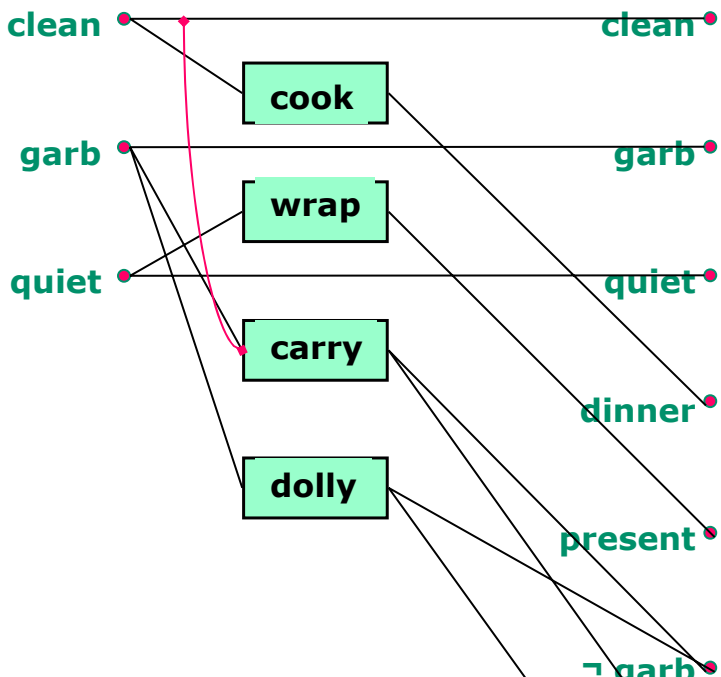
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



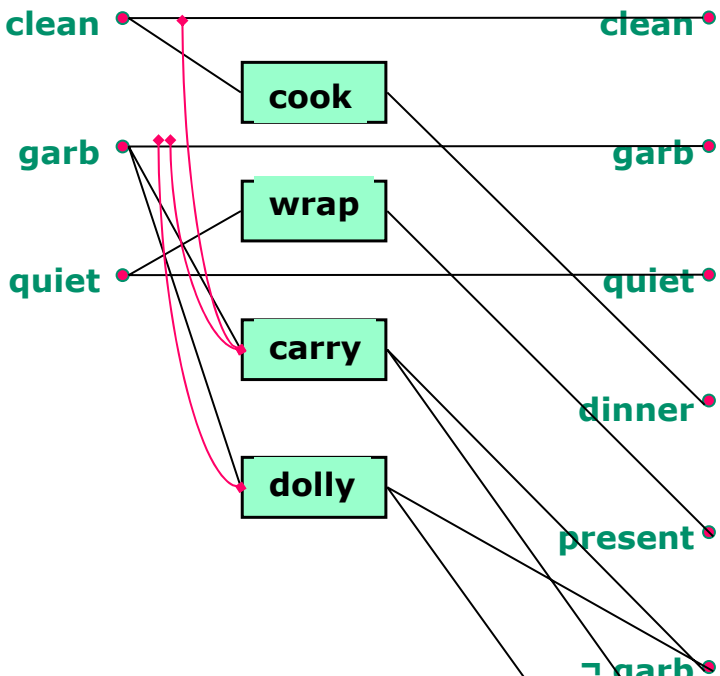
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



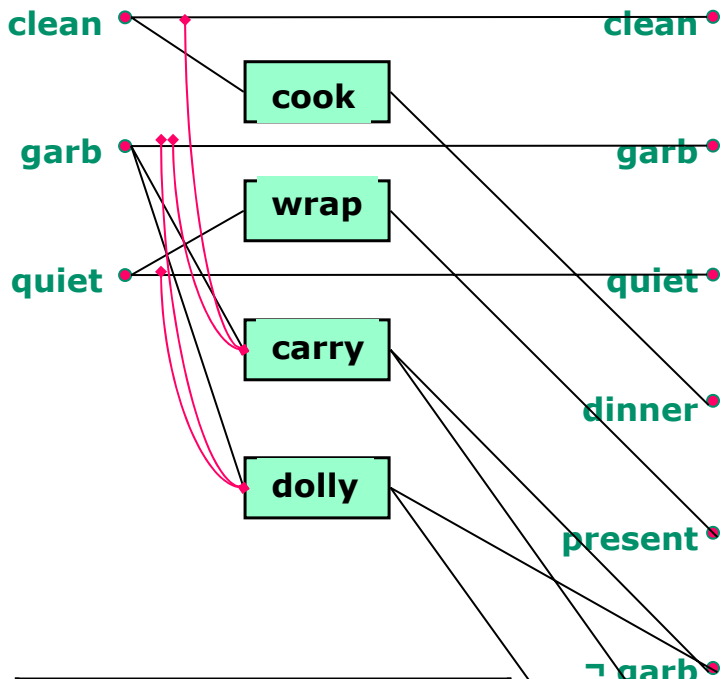
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



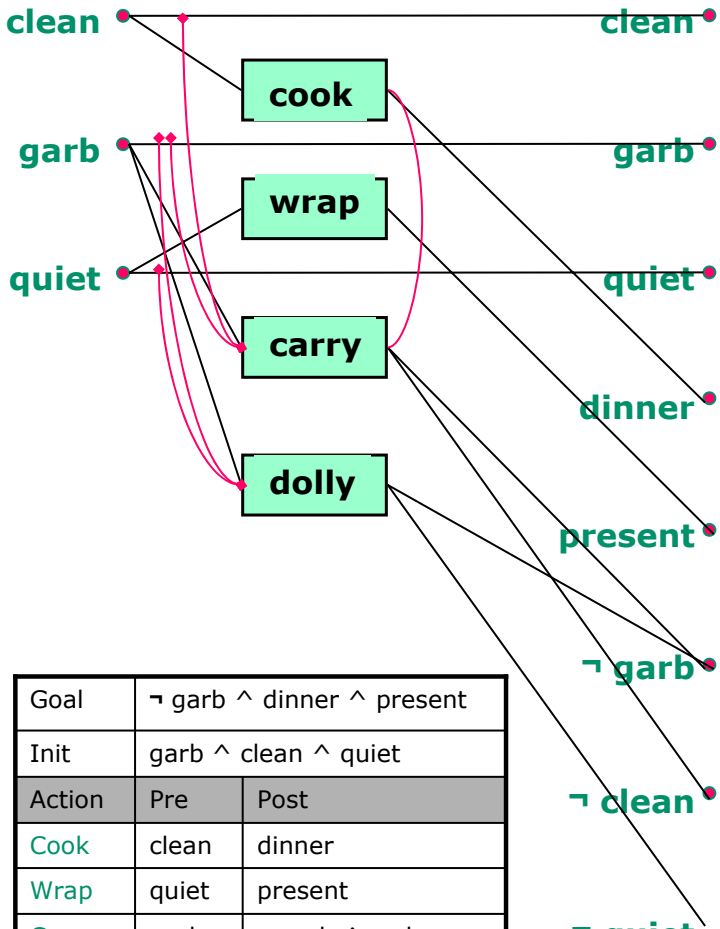
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



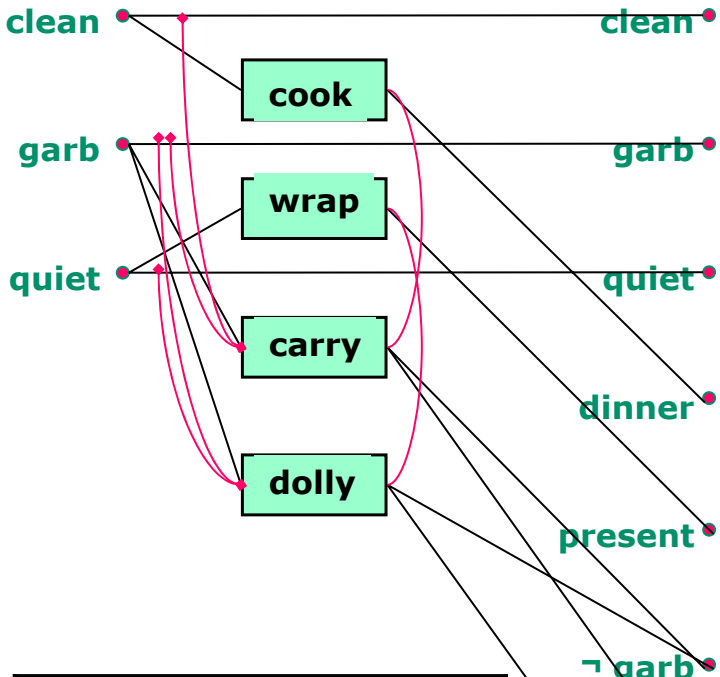
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



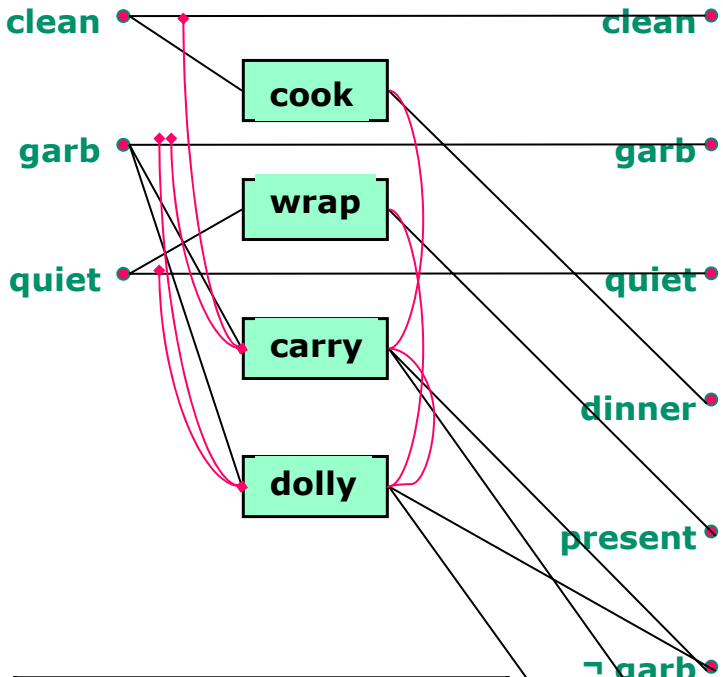
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



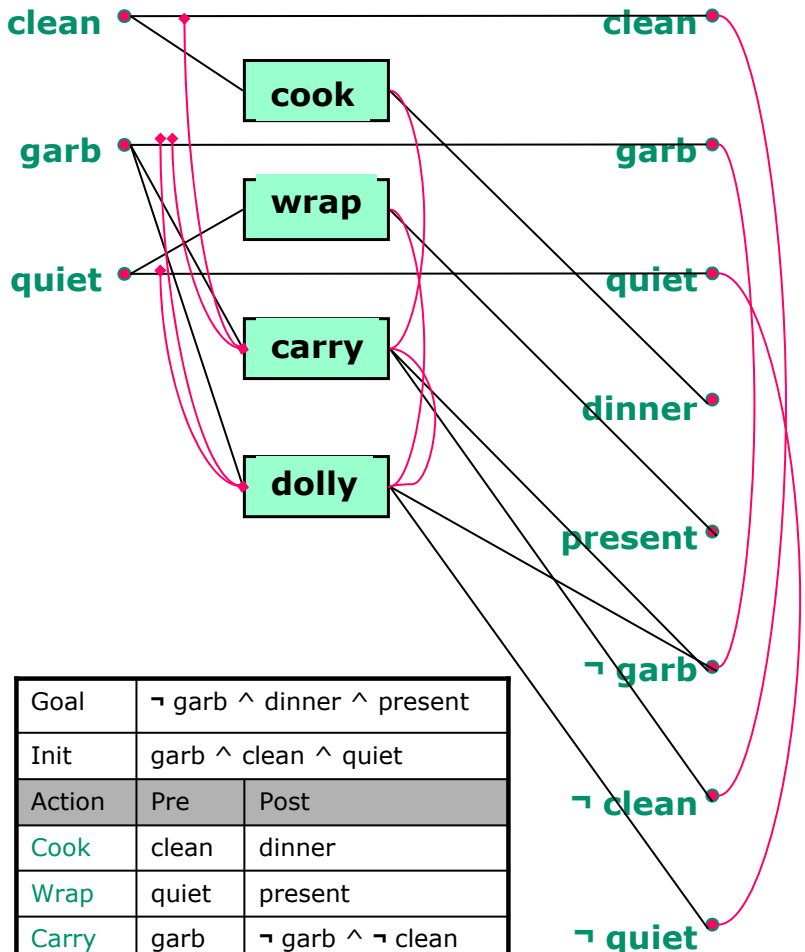
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



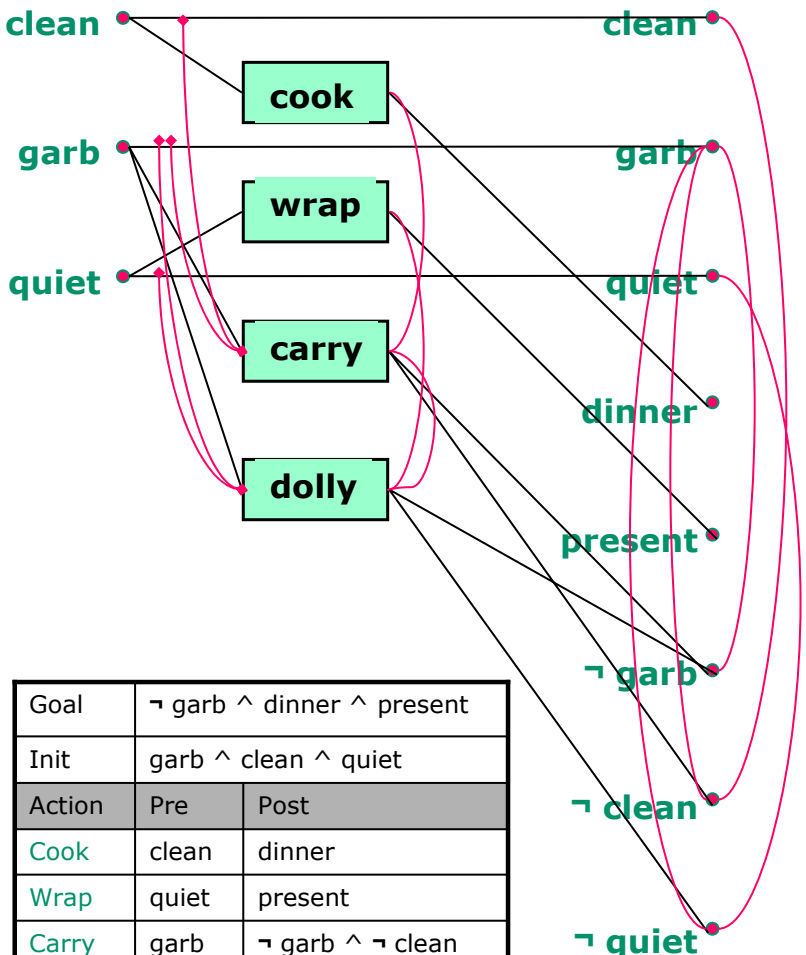
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



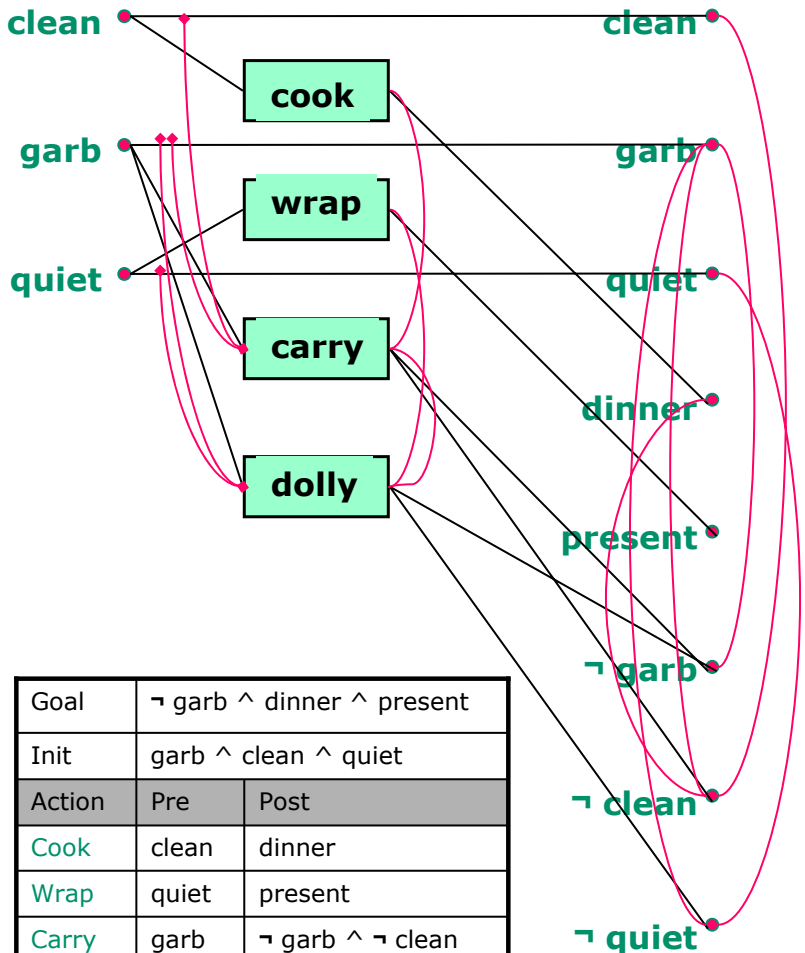
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



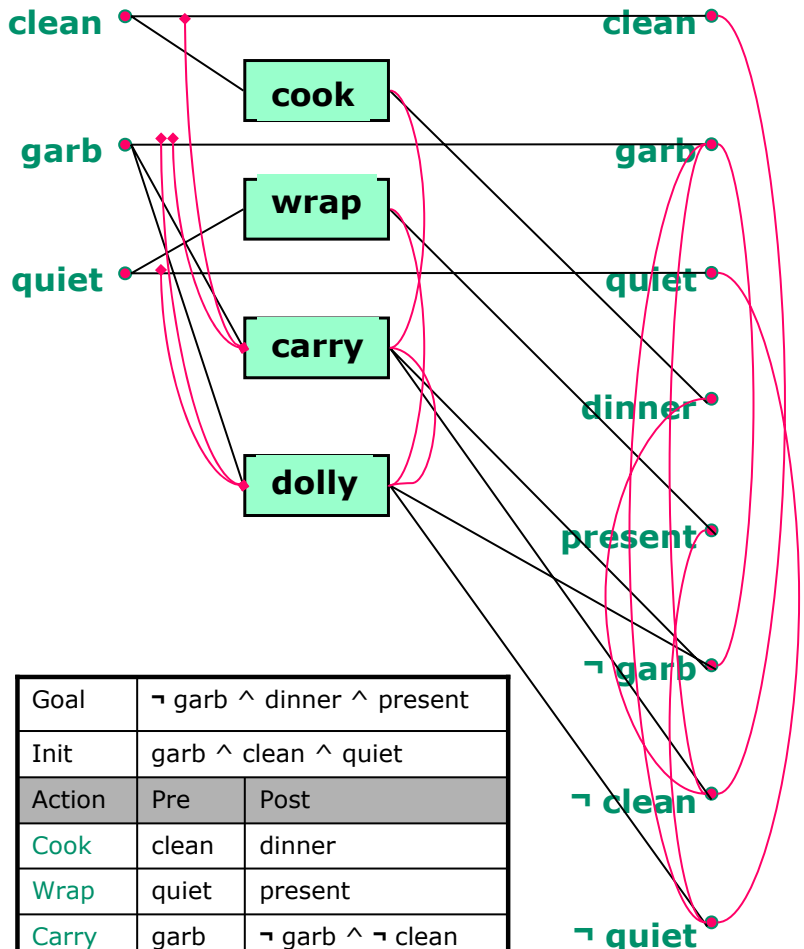
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



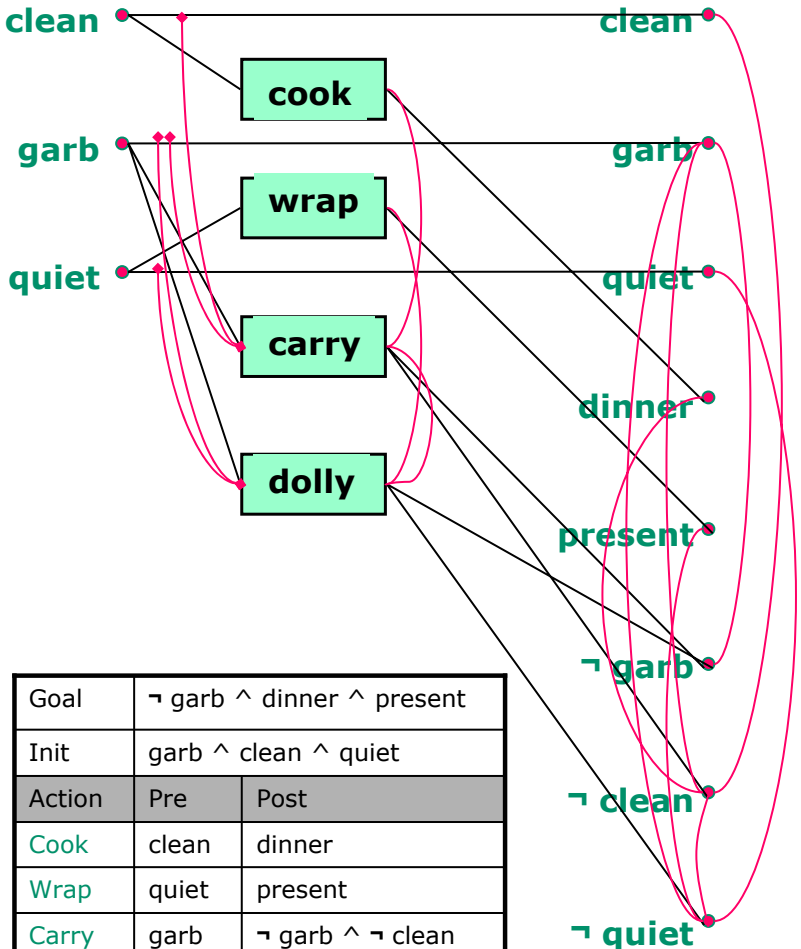
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



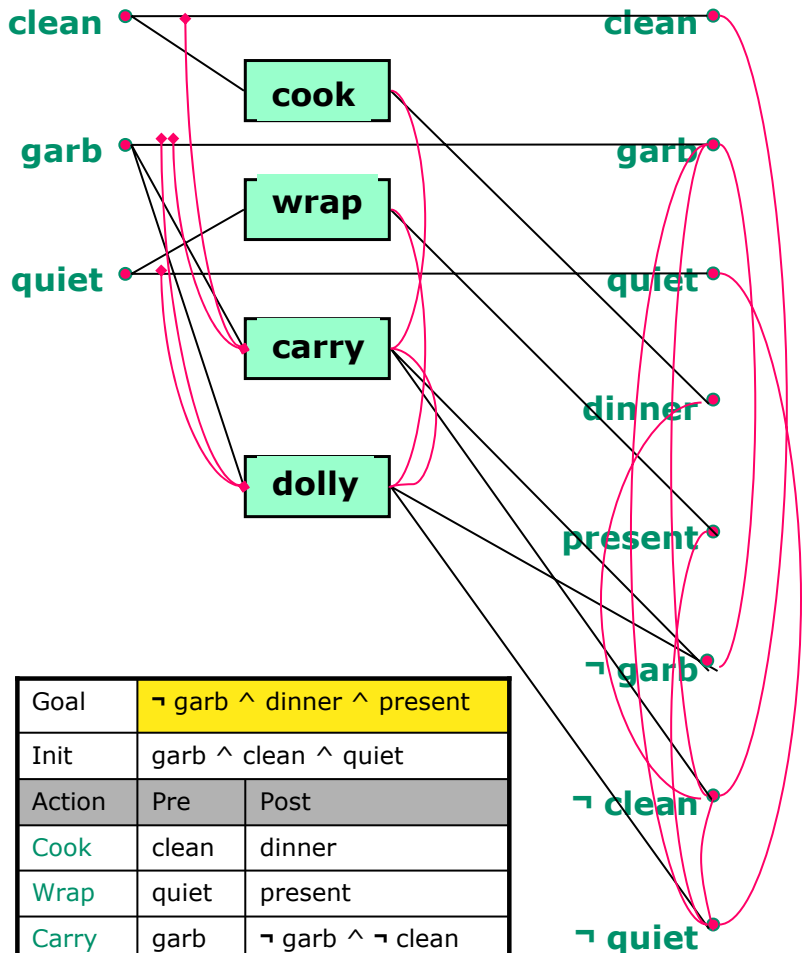
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



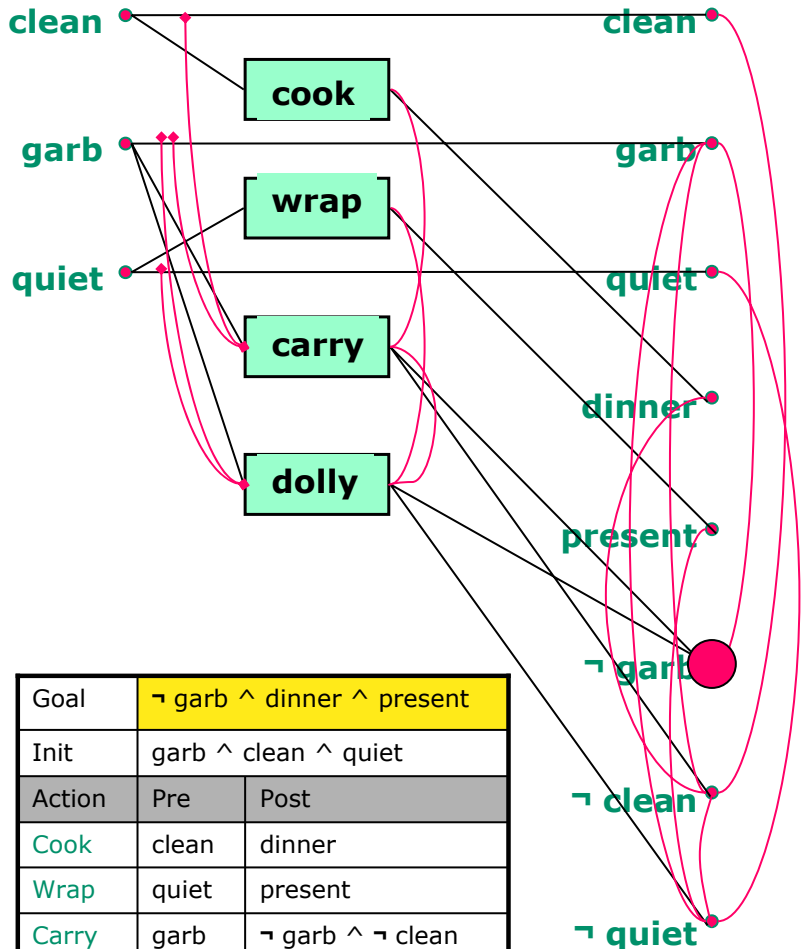
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



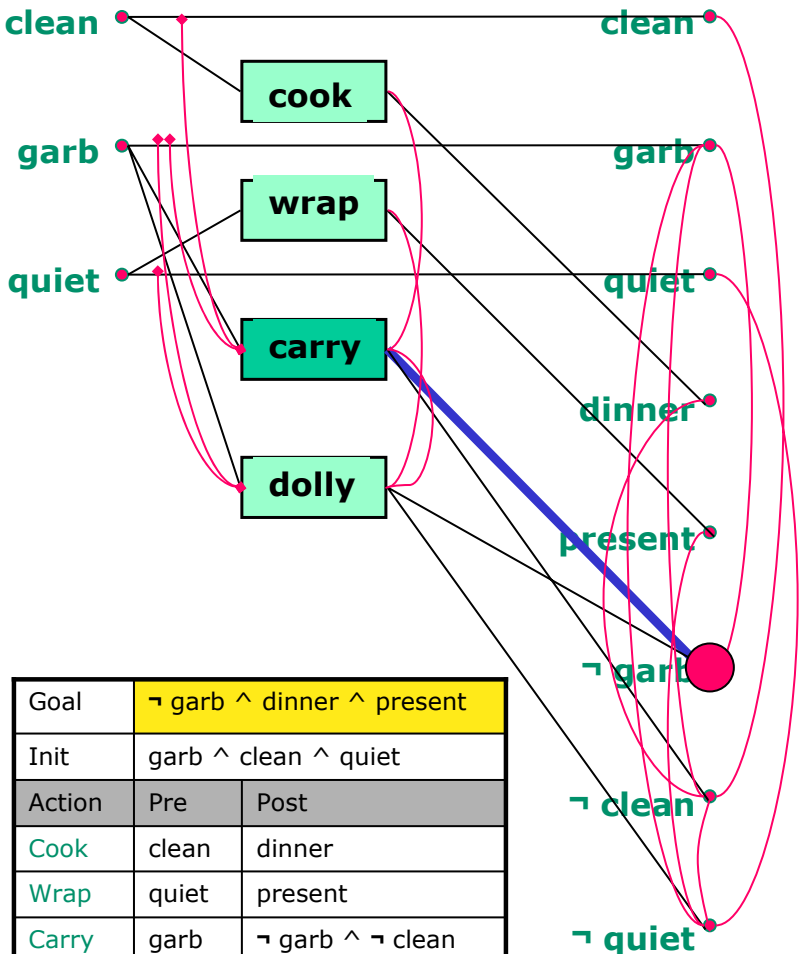
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



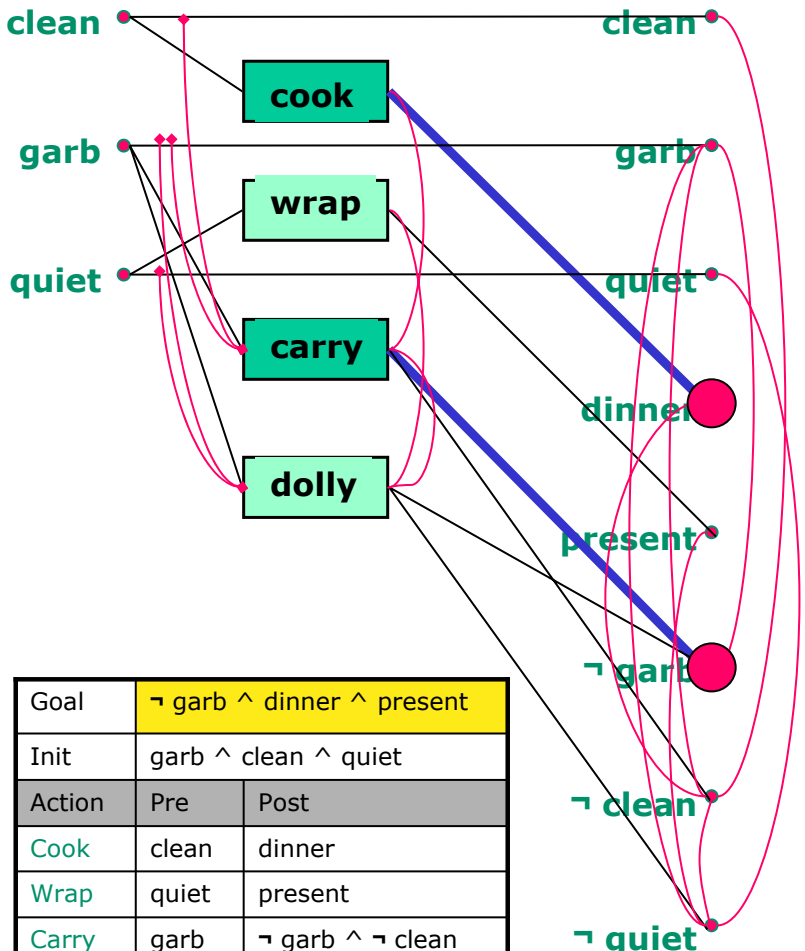
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



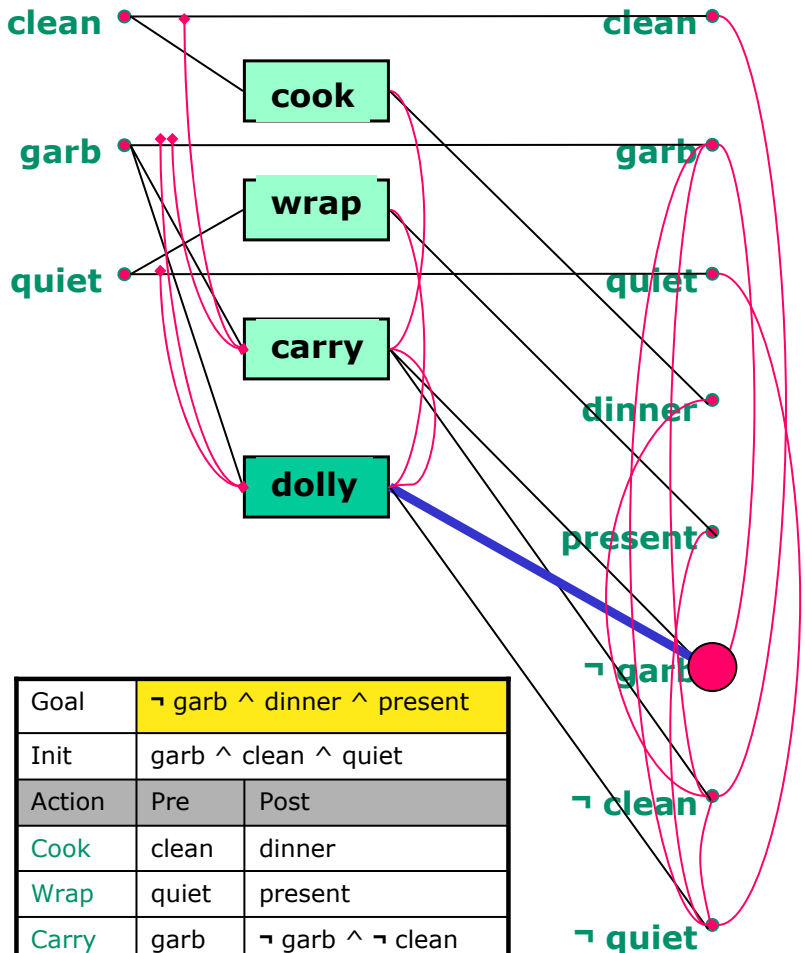
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



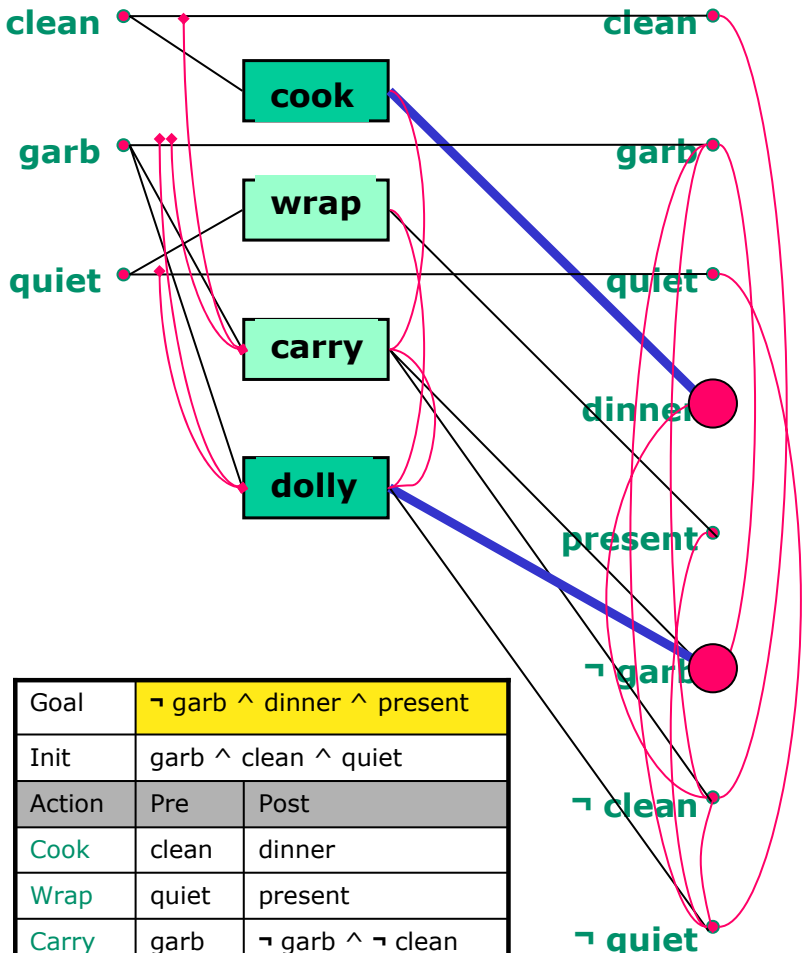
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



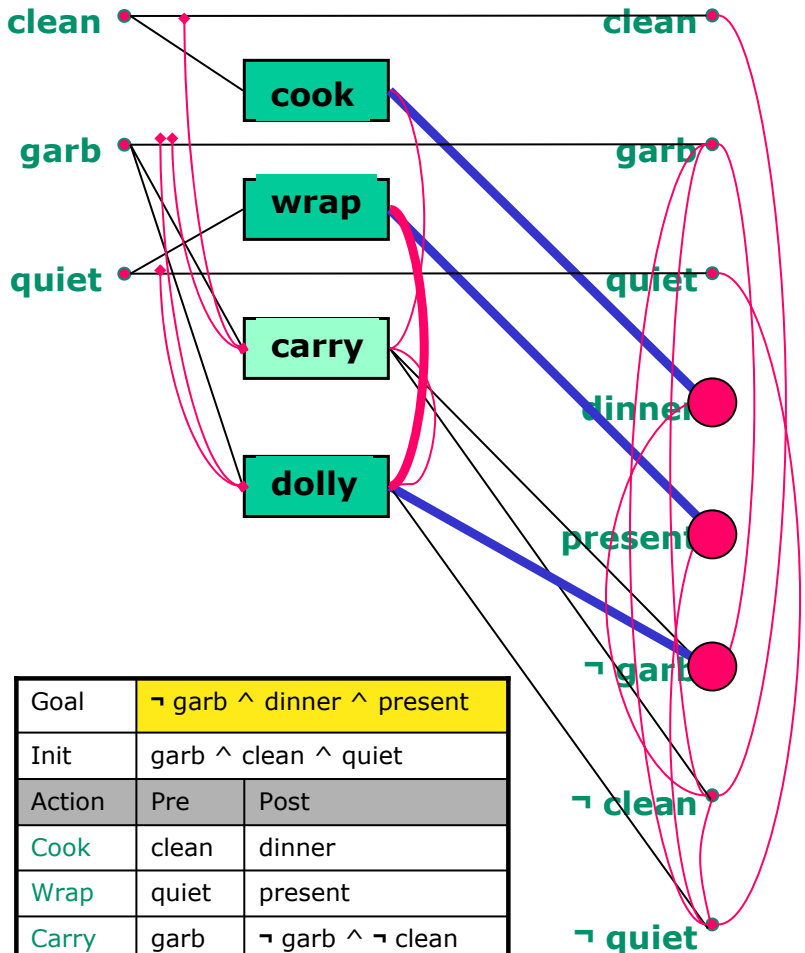
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



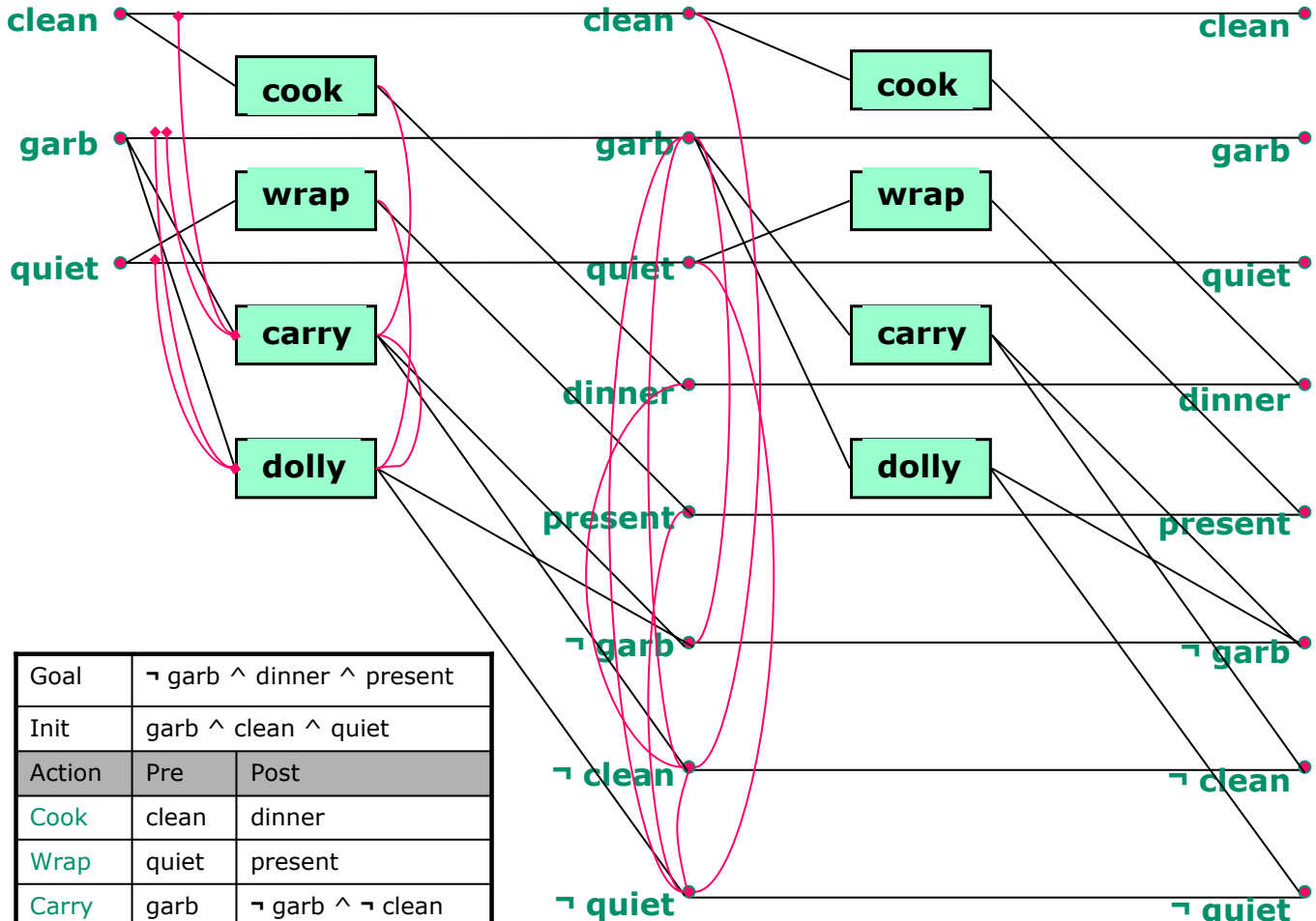
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



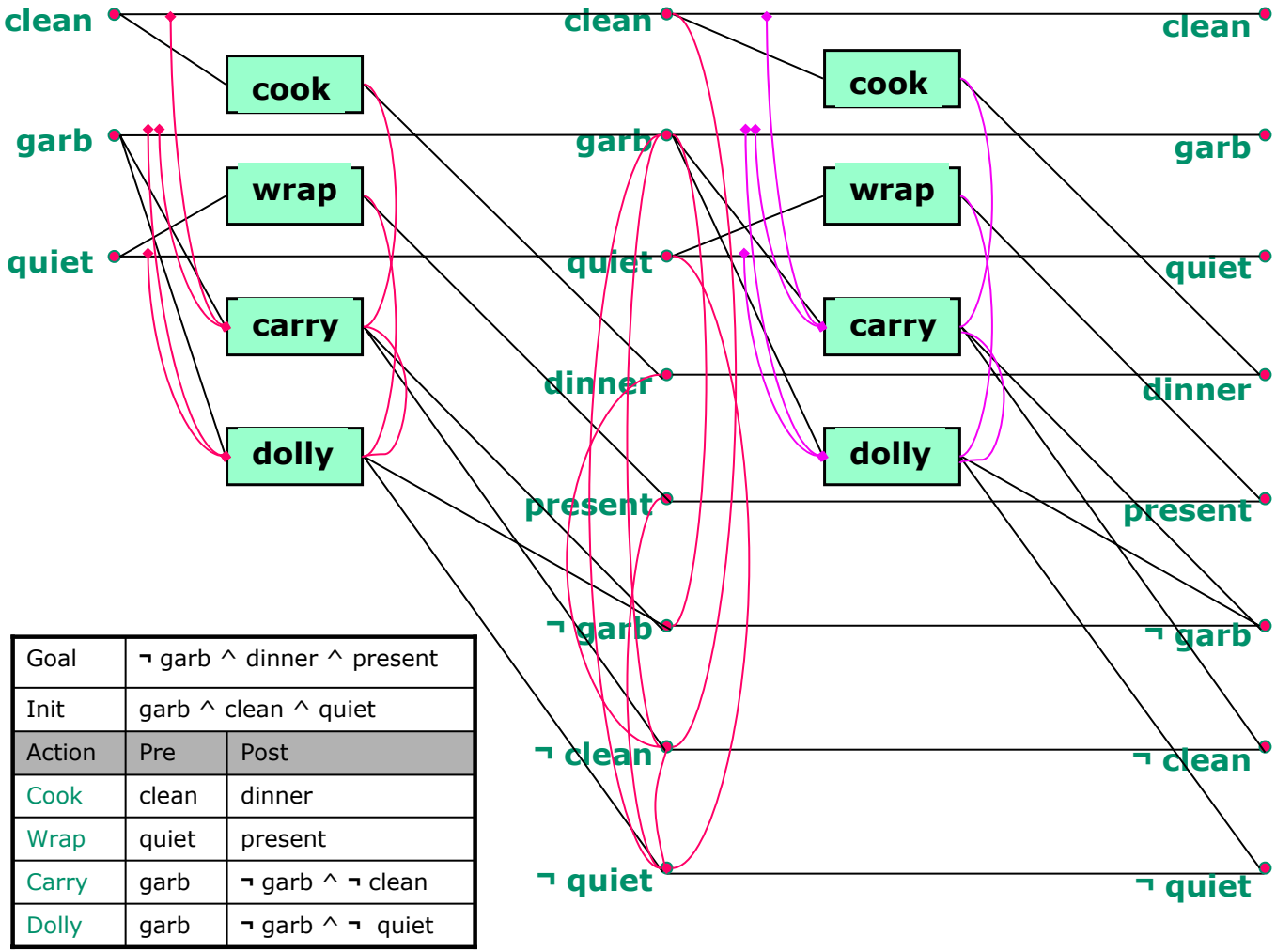
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



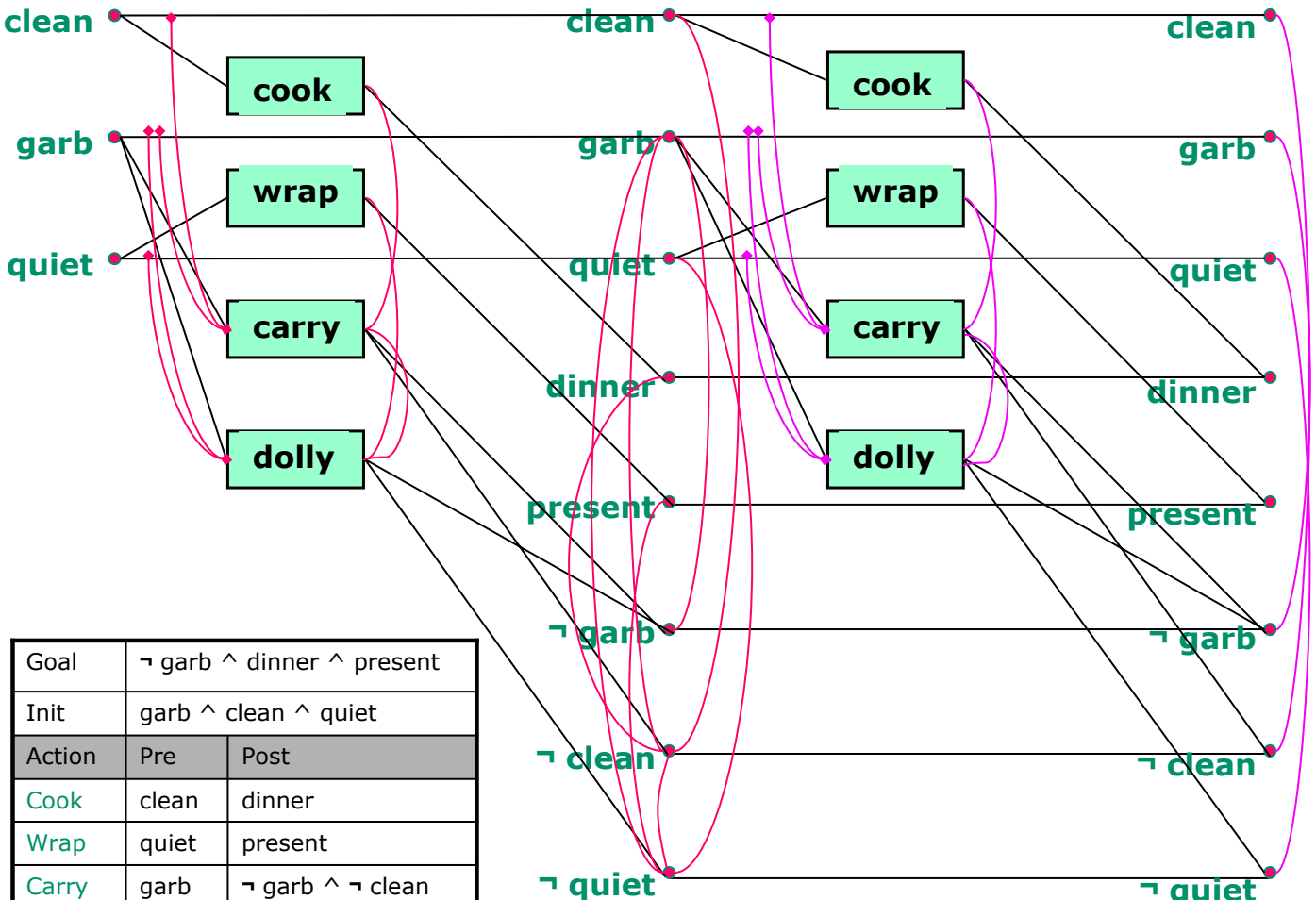
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



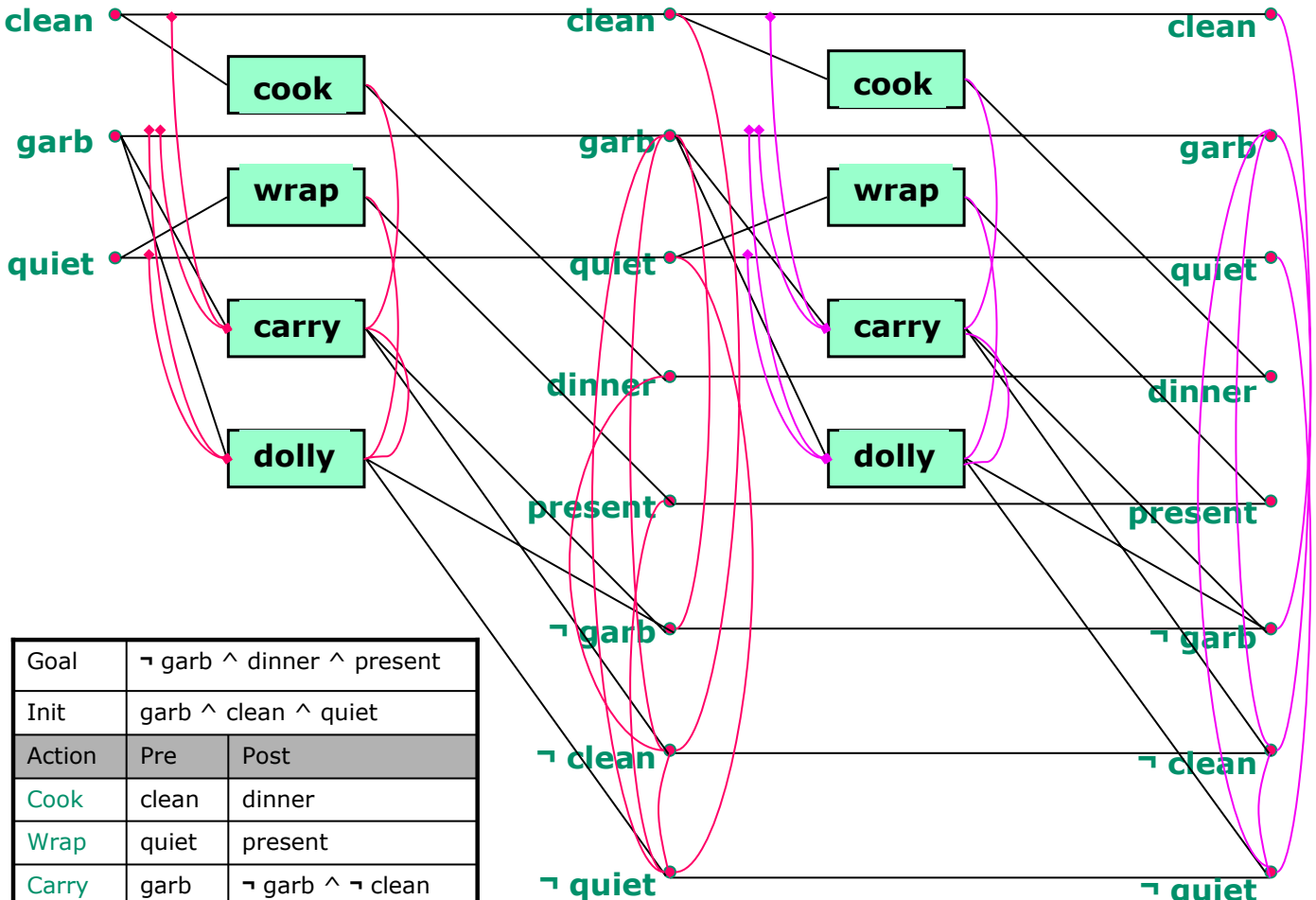
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



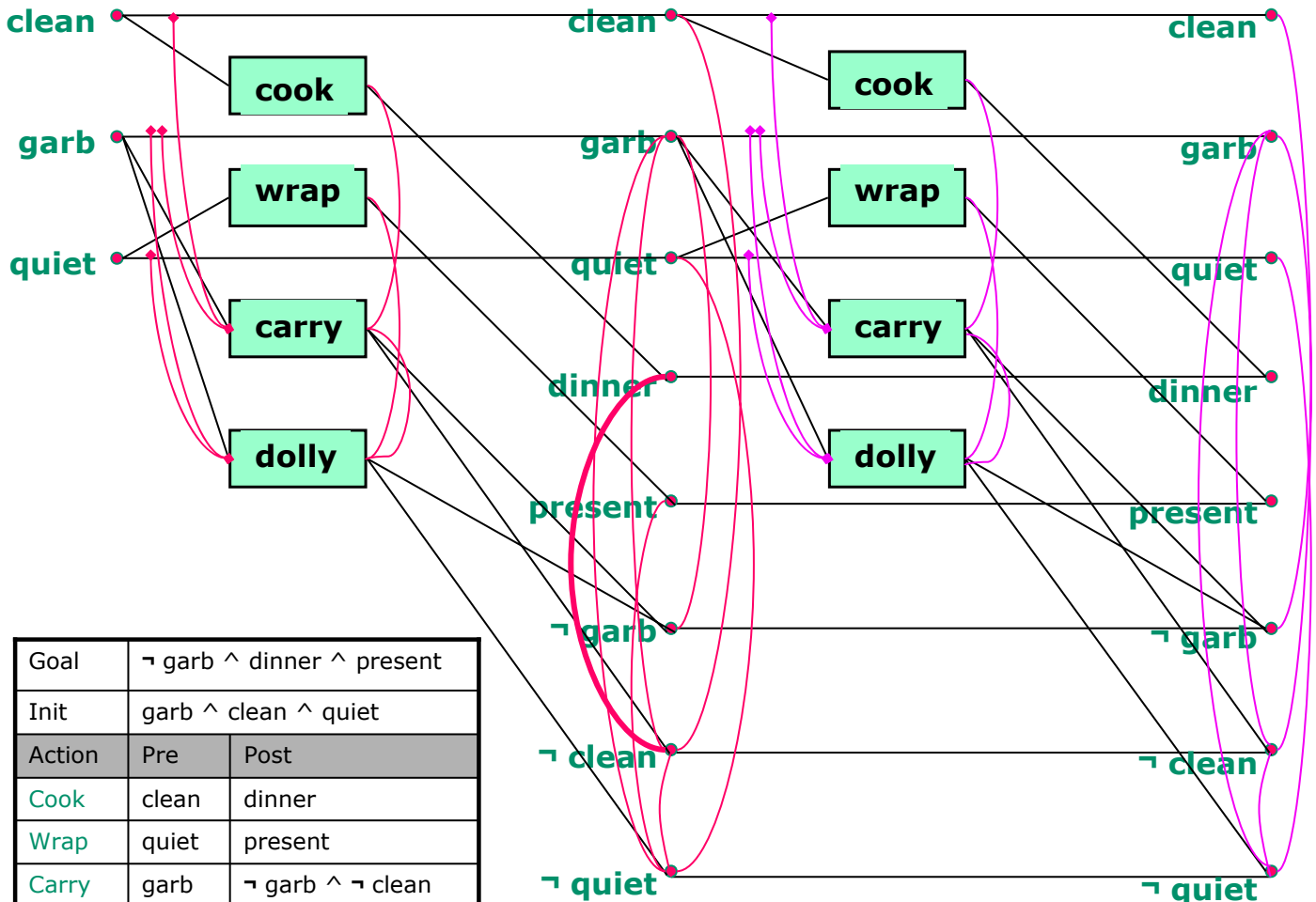
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



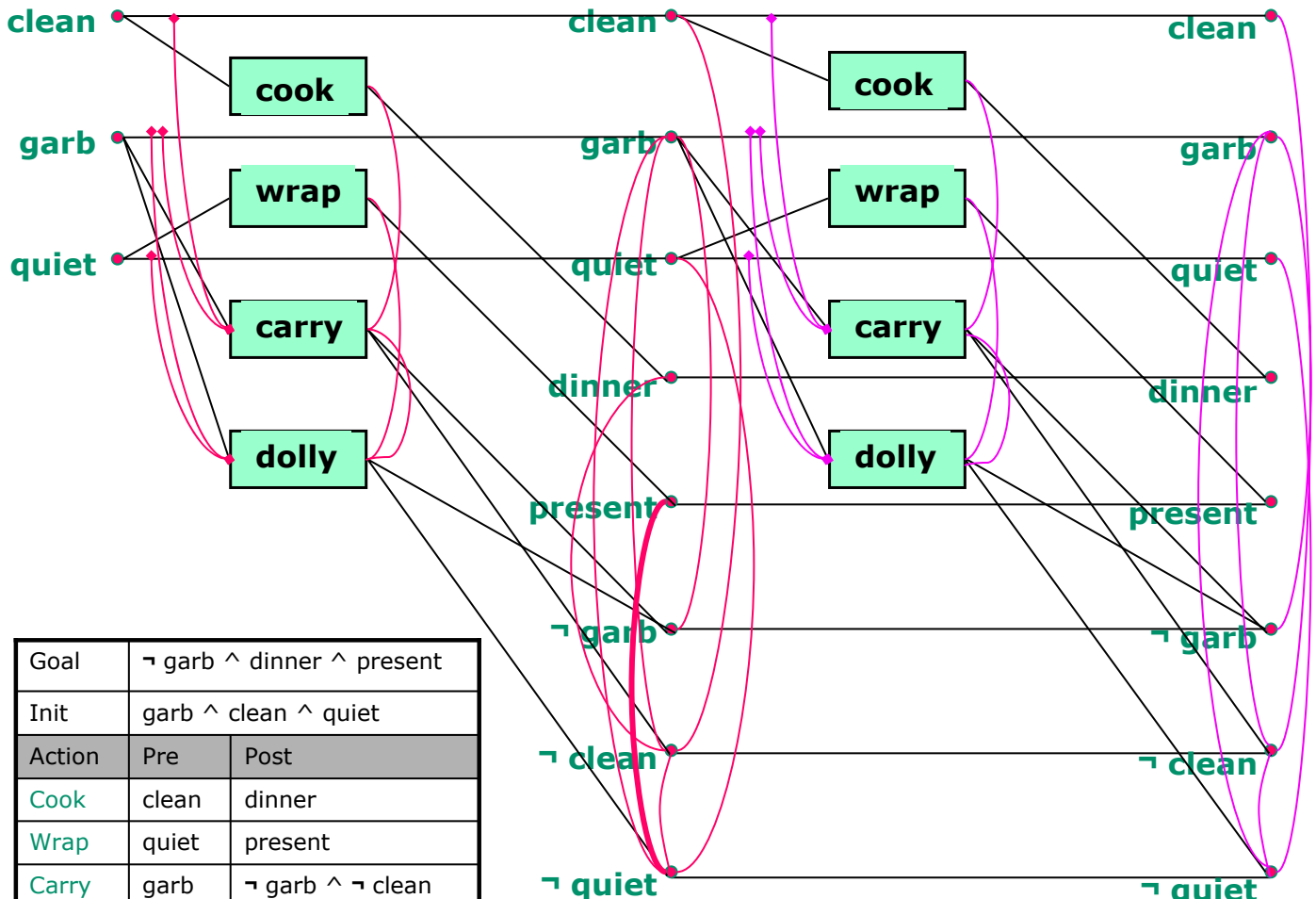
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



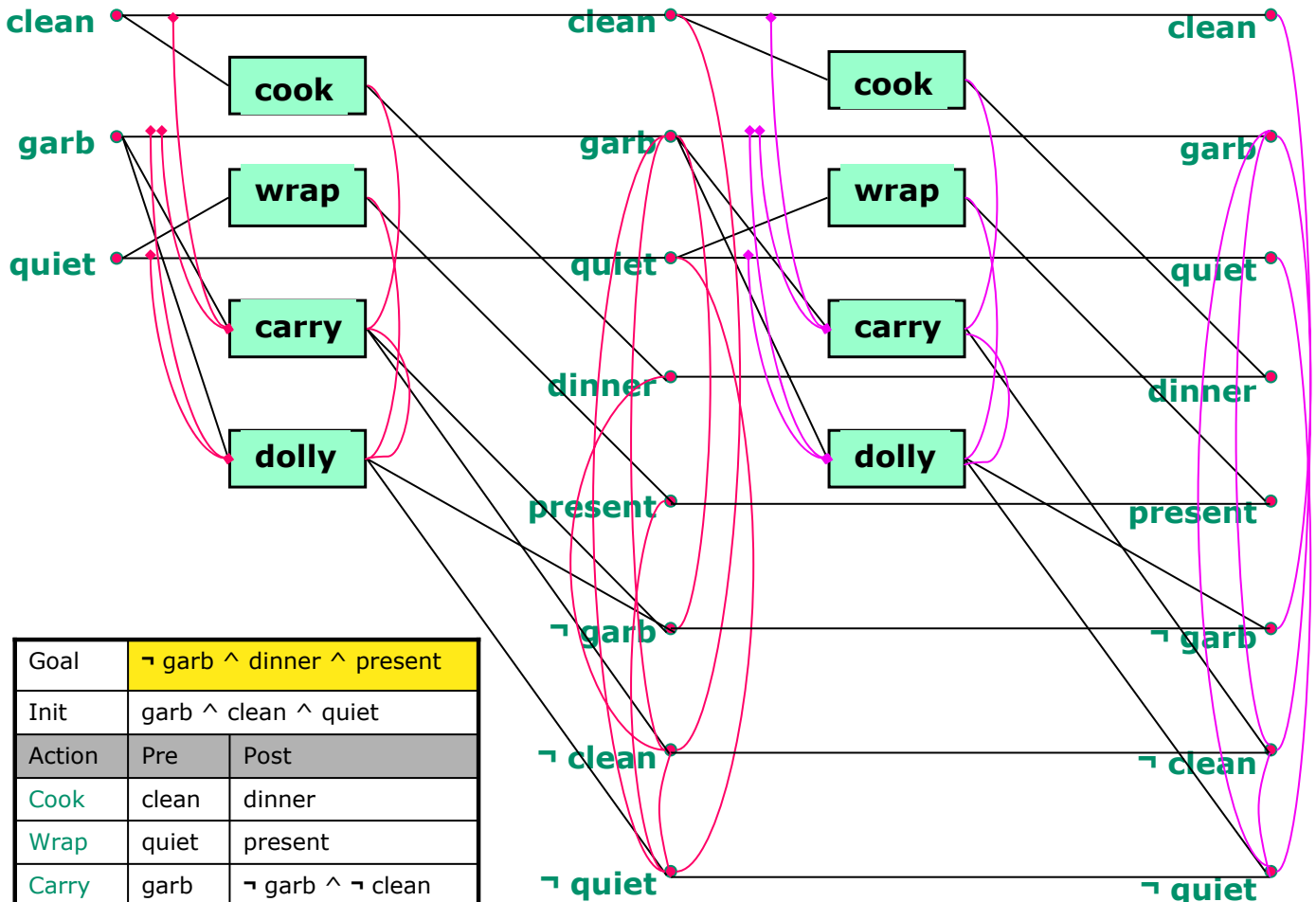
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



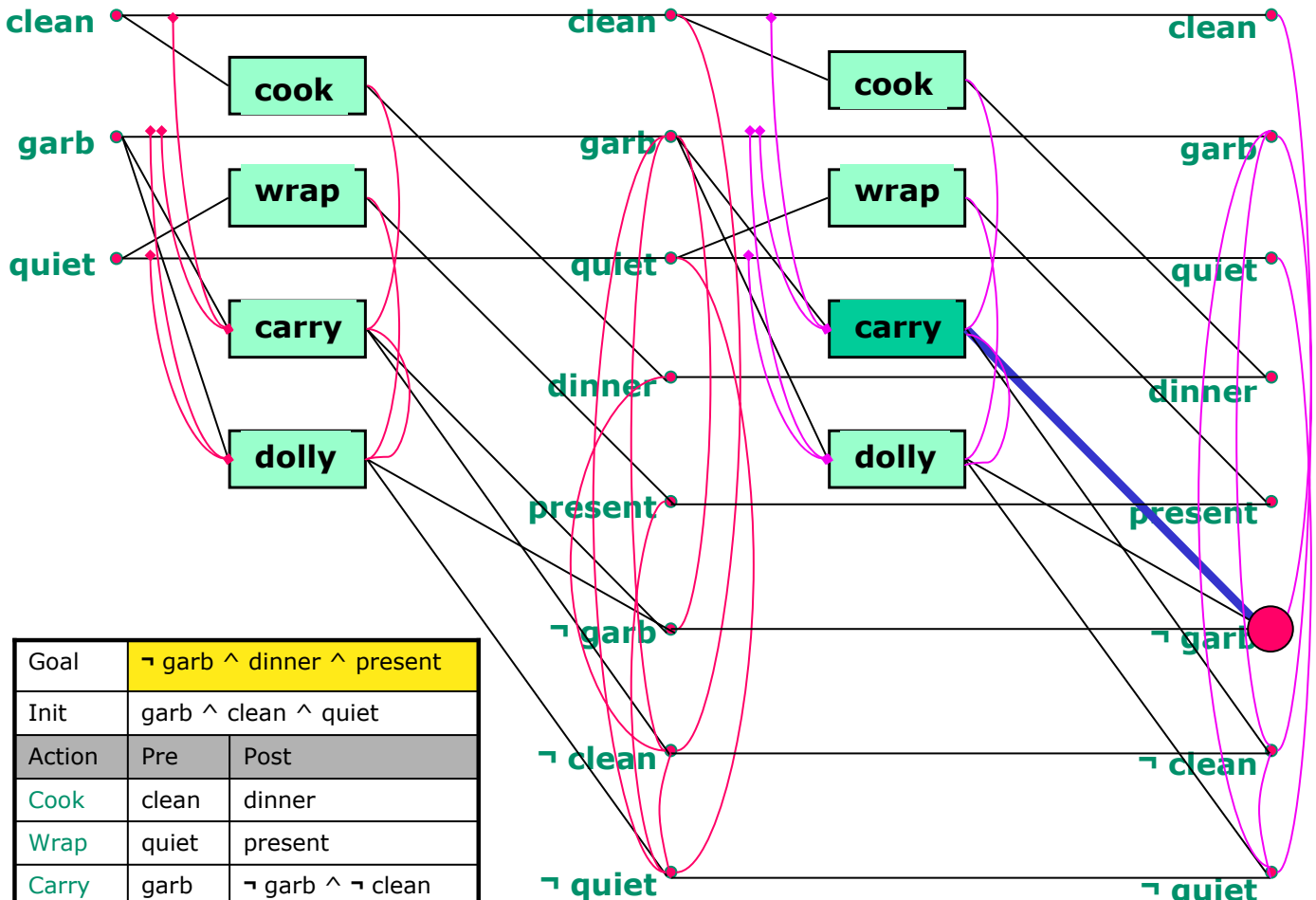
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



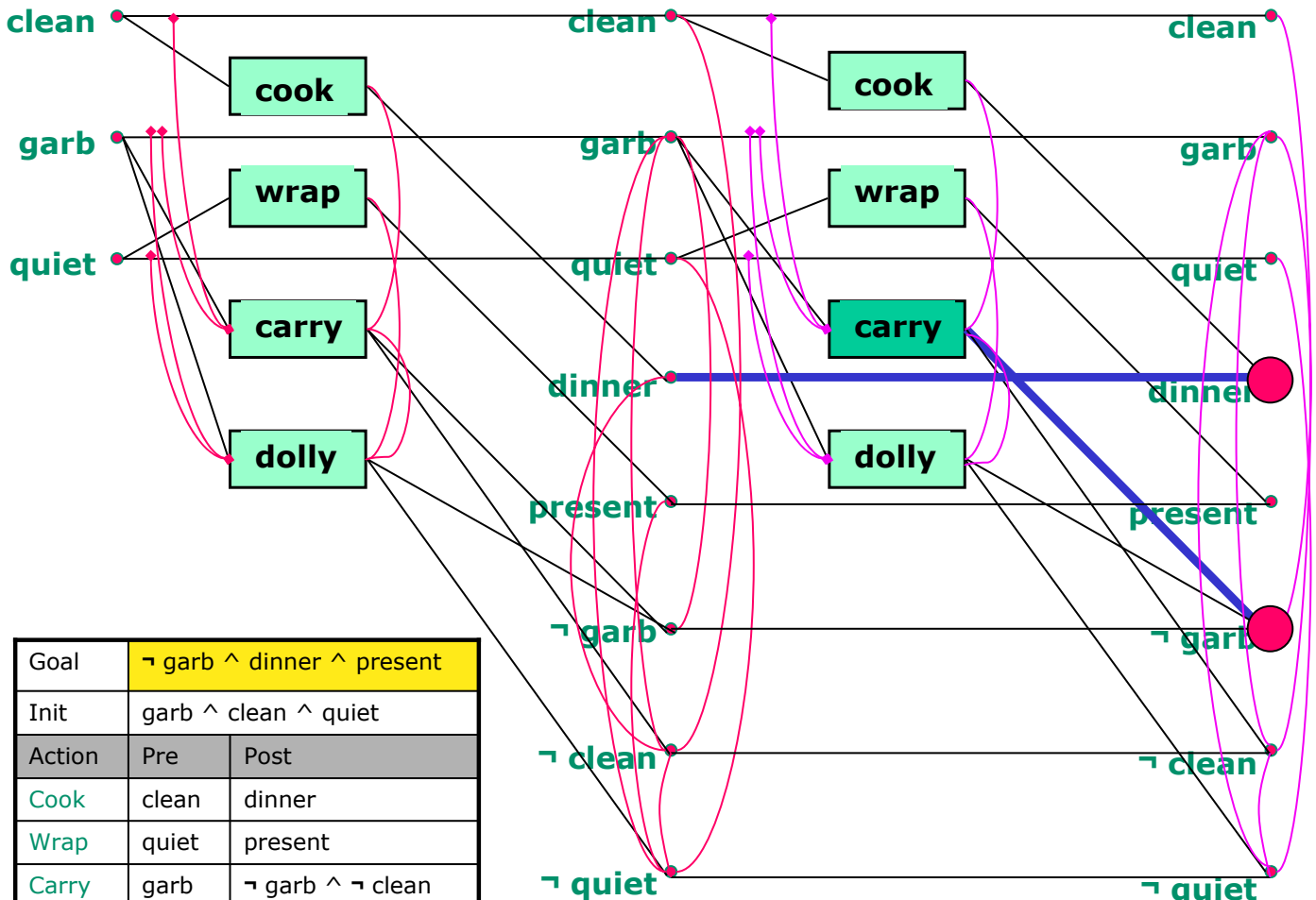
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



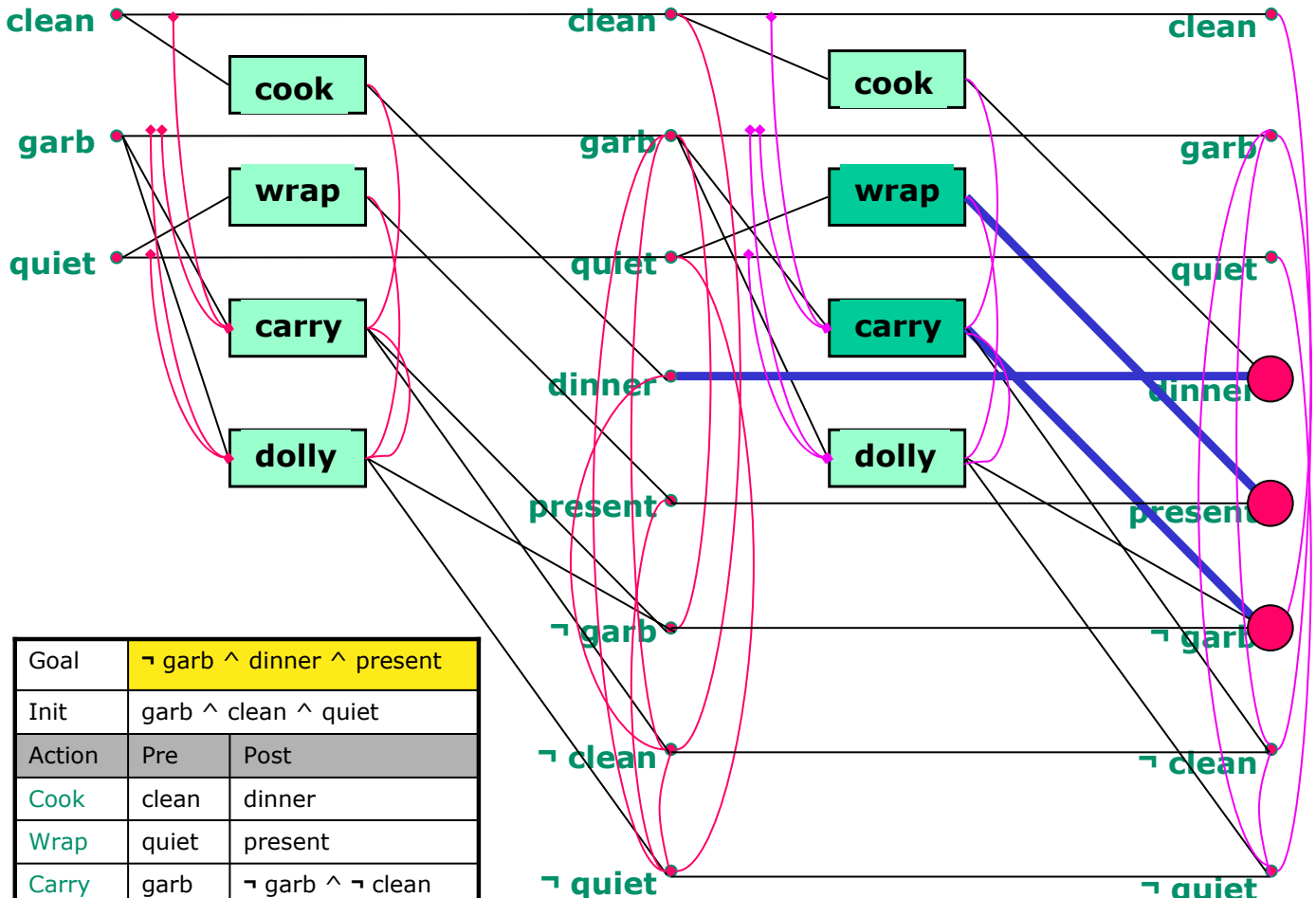
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



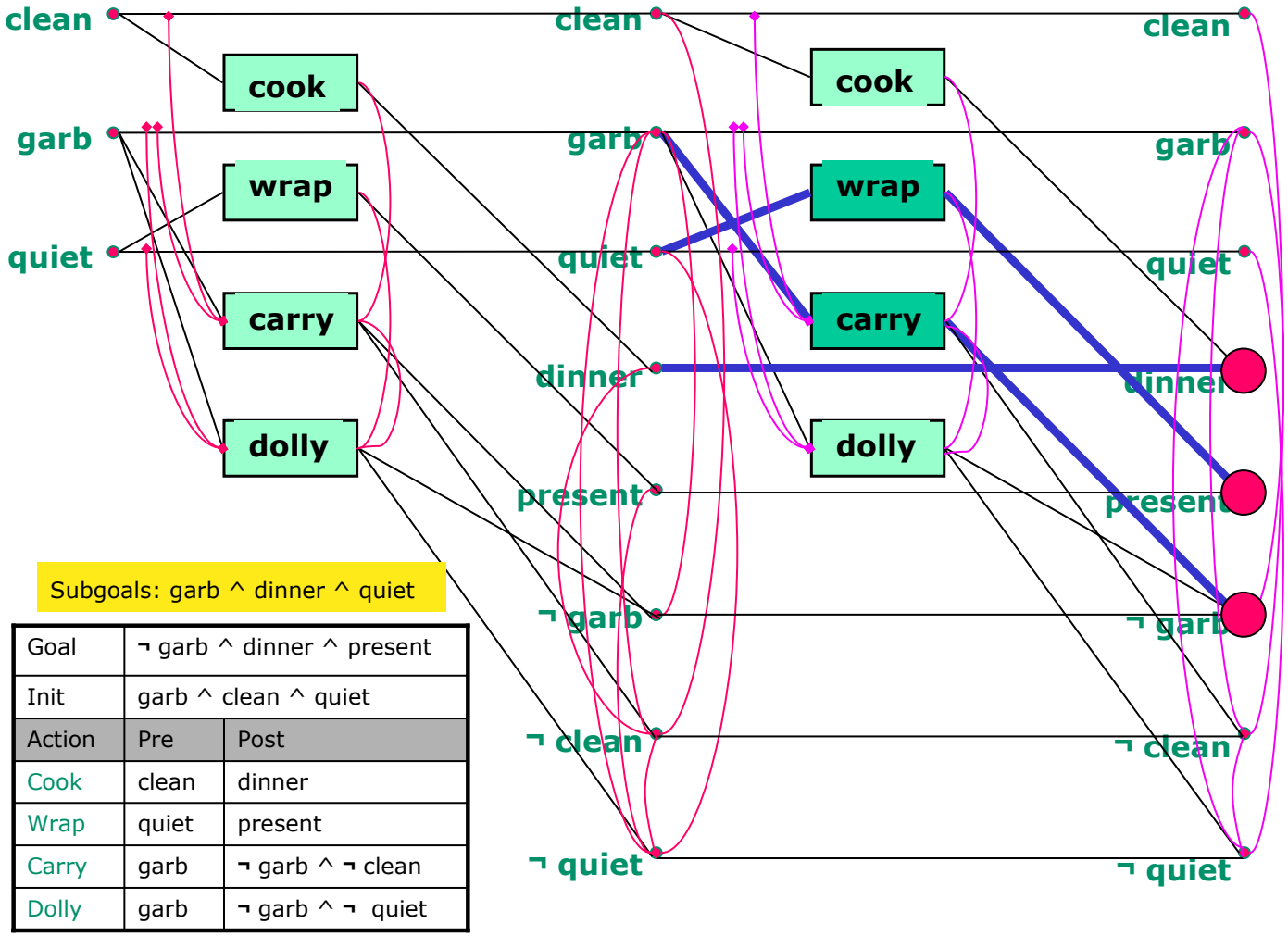
Goal	¬ garb ^ dinner ^ present	
Init	garb ^ clean ^ quiet	
Action	Pre	Post
Cook	clean	dinner
Wrap	quiet	present
Carry	garb	¬ garb ^ ¬ clean
Dolly	garb	¬ garb ^ ¬ quiet

Birthday Dinner example



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

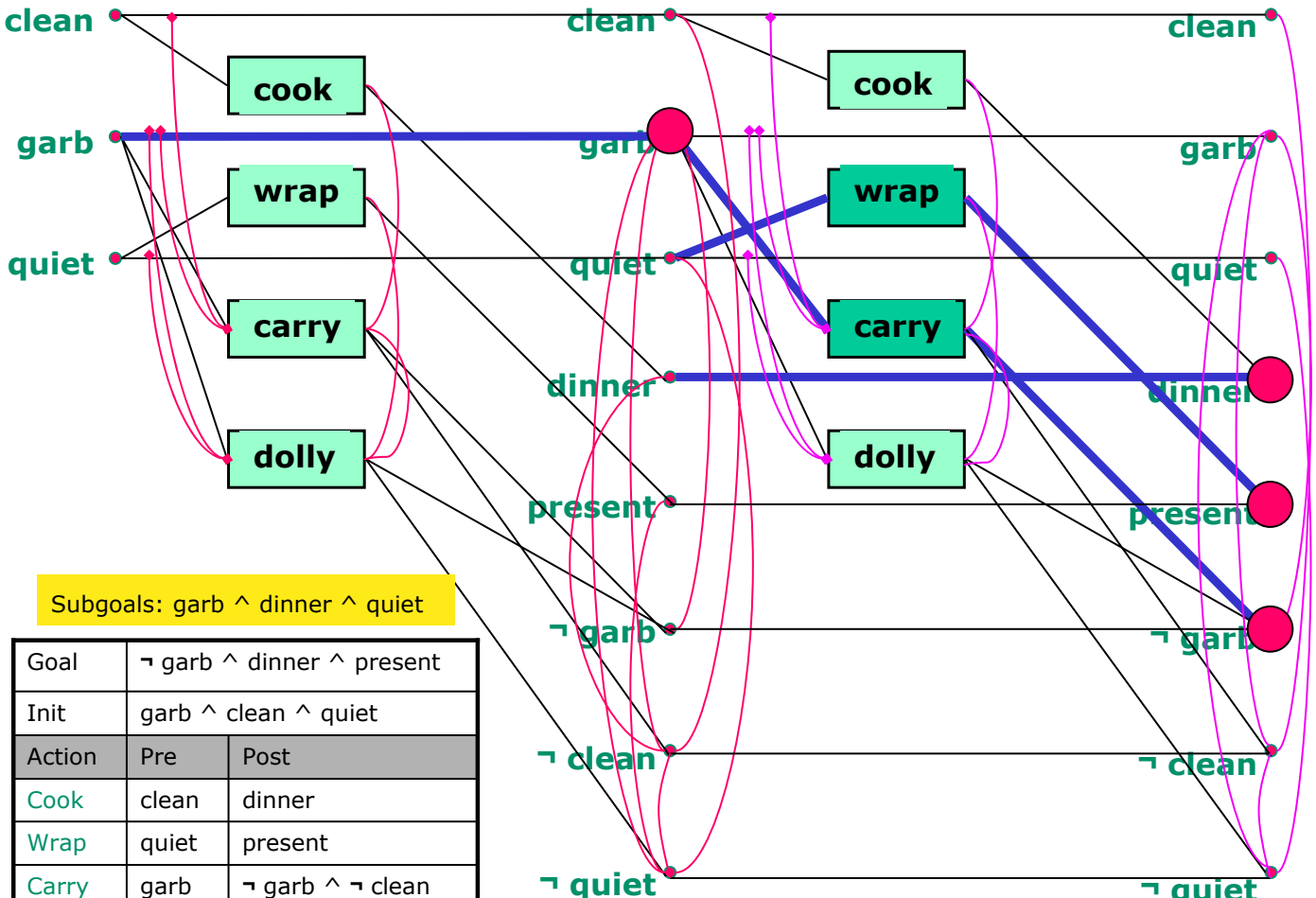
Birthday Dinner example



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

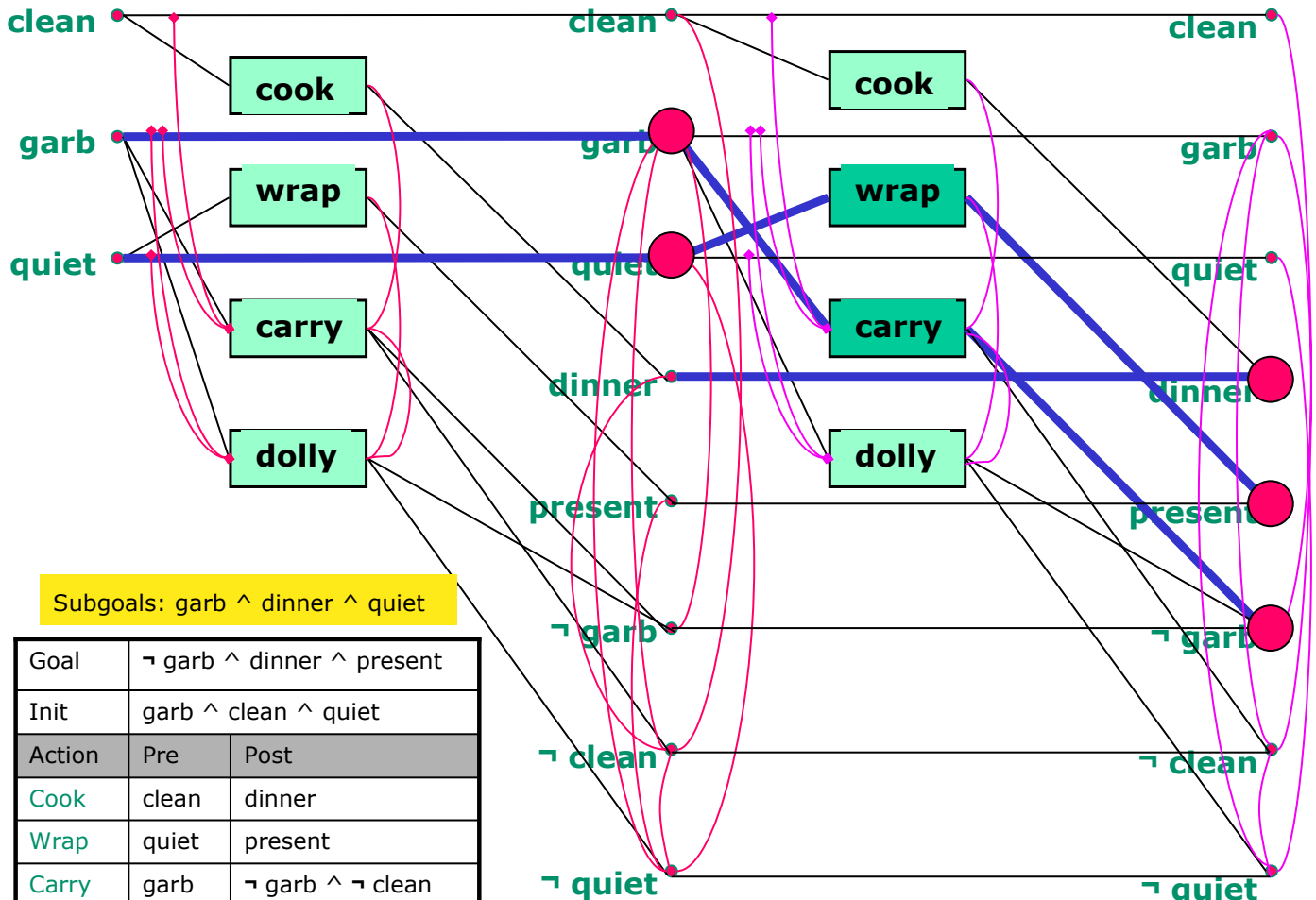
Birthday Dinner example



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

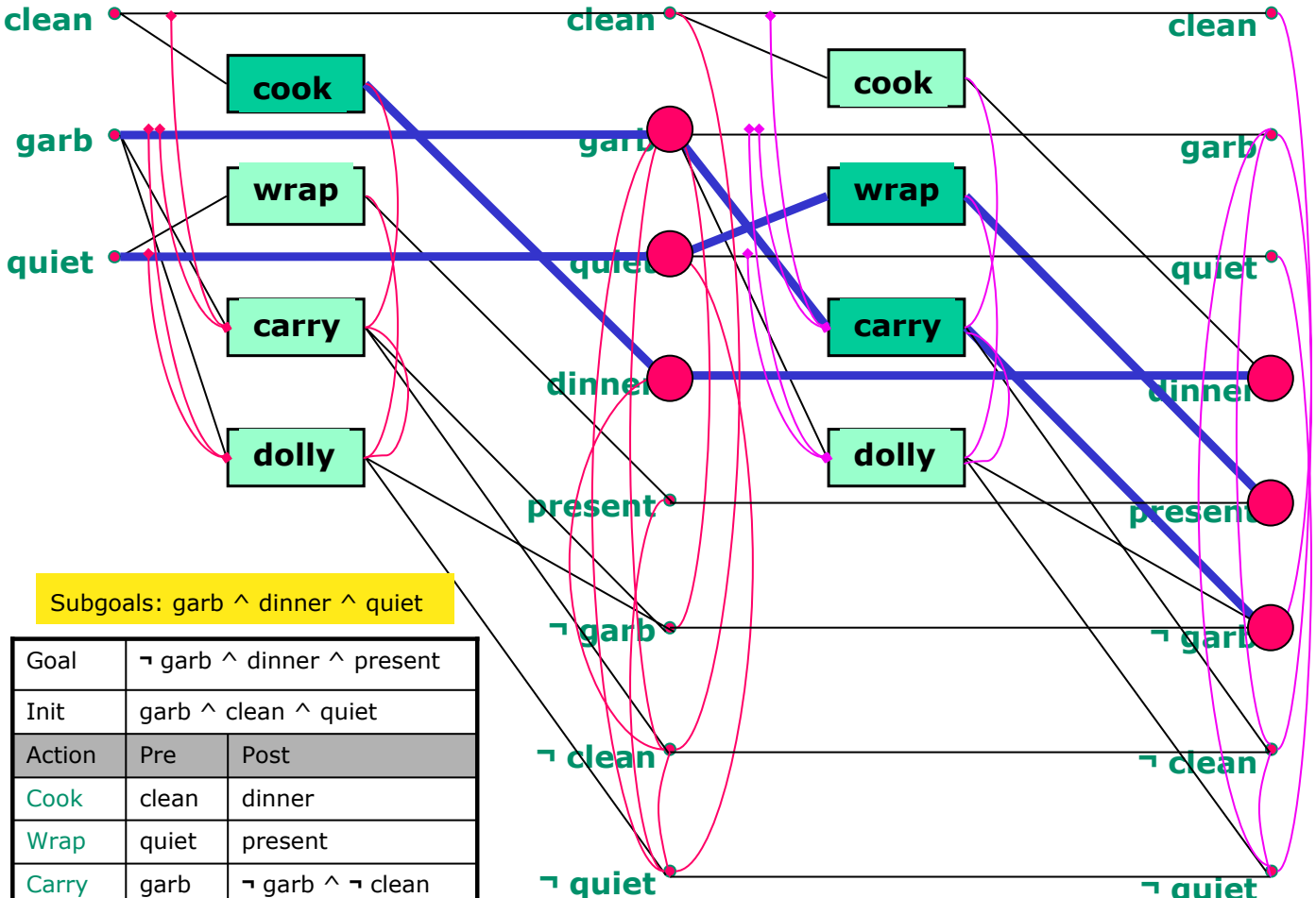
Birthday Dinner example



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

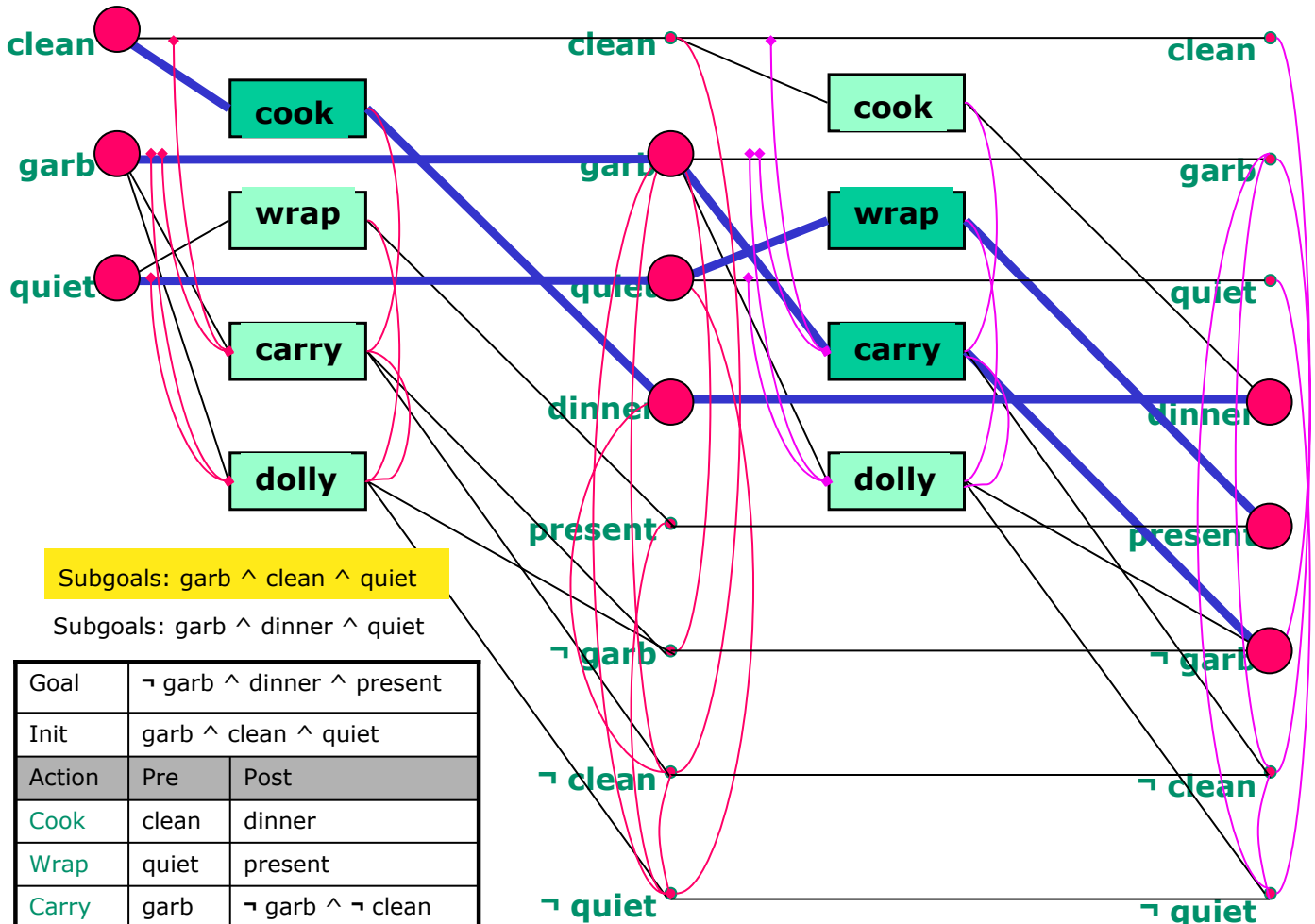
Birthday Dinner example



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

Birthday Dinner example



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

Zusammenfassung

- Planungsprobleme sind spezielle Suchprobleme mit *faktorisierter* Zustandsrepräsentation. Sie können in PDDL formalisiert werden
- Diese Formalisierung erlaubt die Nutzung spezialisierter *Planungsalgorithmen*, die oft um Größenordnungen schneller als allgemeine Suchalgorithmen sind
- Außerdem ermöglicht diese Formalisierung die Definition von guten Heuristiken, sodass die Algorithmen nicht den gesamten Suchraum explorieren müssen
- Partially ordered planning ist ein Planungsalgorithmus, der einen partiell geordneten Plan durch schrittweises Hinzufügen weiterer Aktionen erzeugt
- Graphplan ist ein propositionaler Planungsalgorithmus, der iterativ einen Plan-Graph wachsender Größe erzeugt und aus diesem einen Plan extrahiert