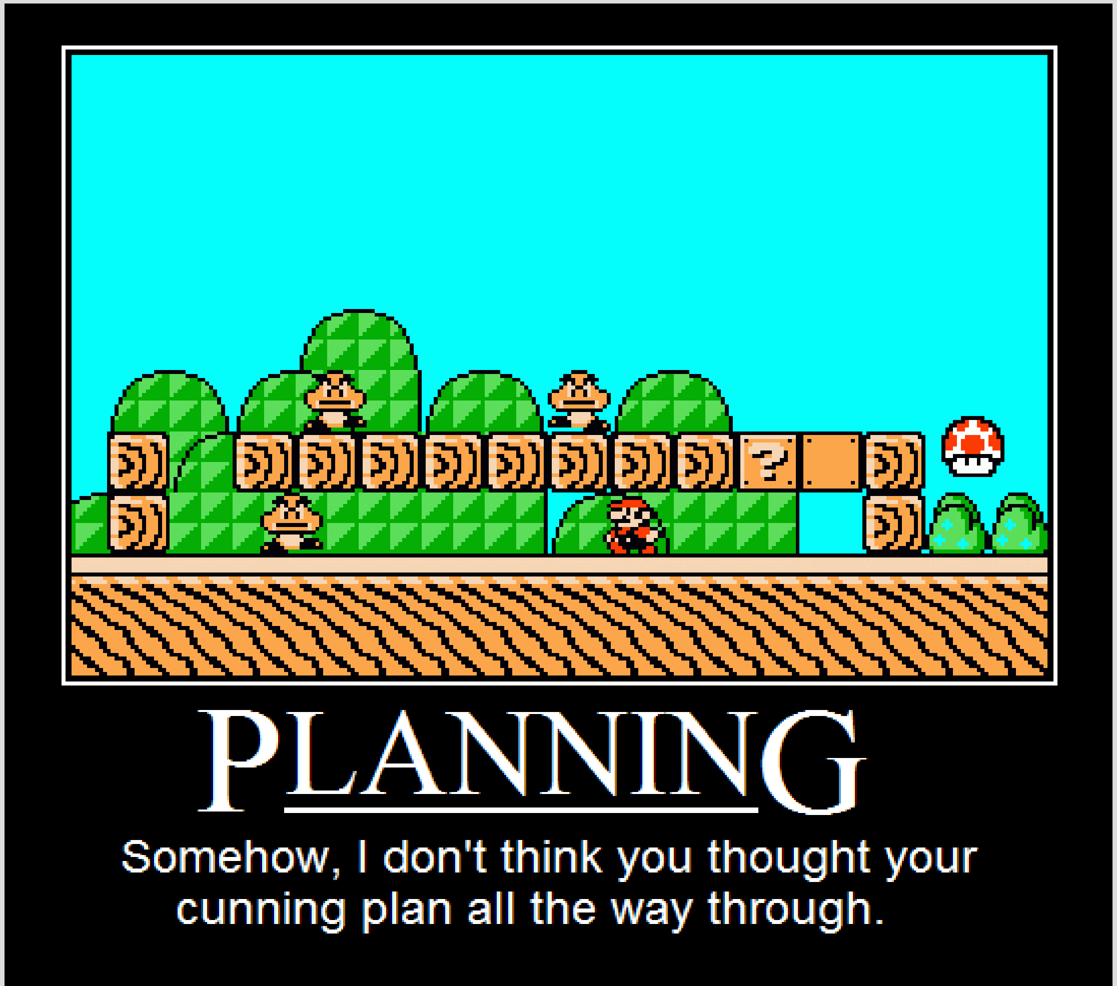
Planning (Ch. 10)


PLANNING
Somehow, I don't think you thought your cunning plan all the way through.

Graph Plan

Consider this problem:

Initial: $Sleepy(me) \wedge Hungry(me)$

Goal: $\neg Sleepy(me) \wedge \neg Hungry(me)$

Action($Eat(x)$,
Precondition: $Hungry(x)$,
Effect: $\neg Hungry(x)$)

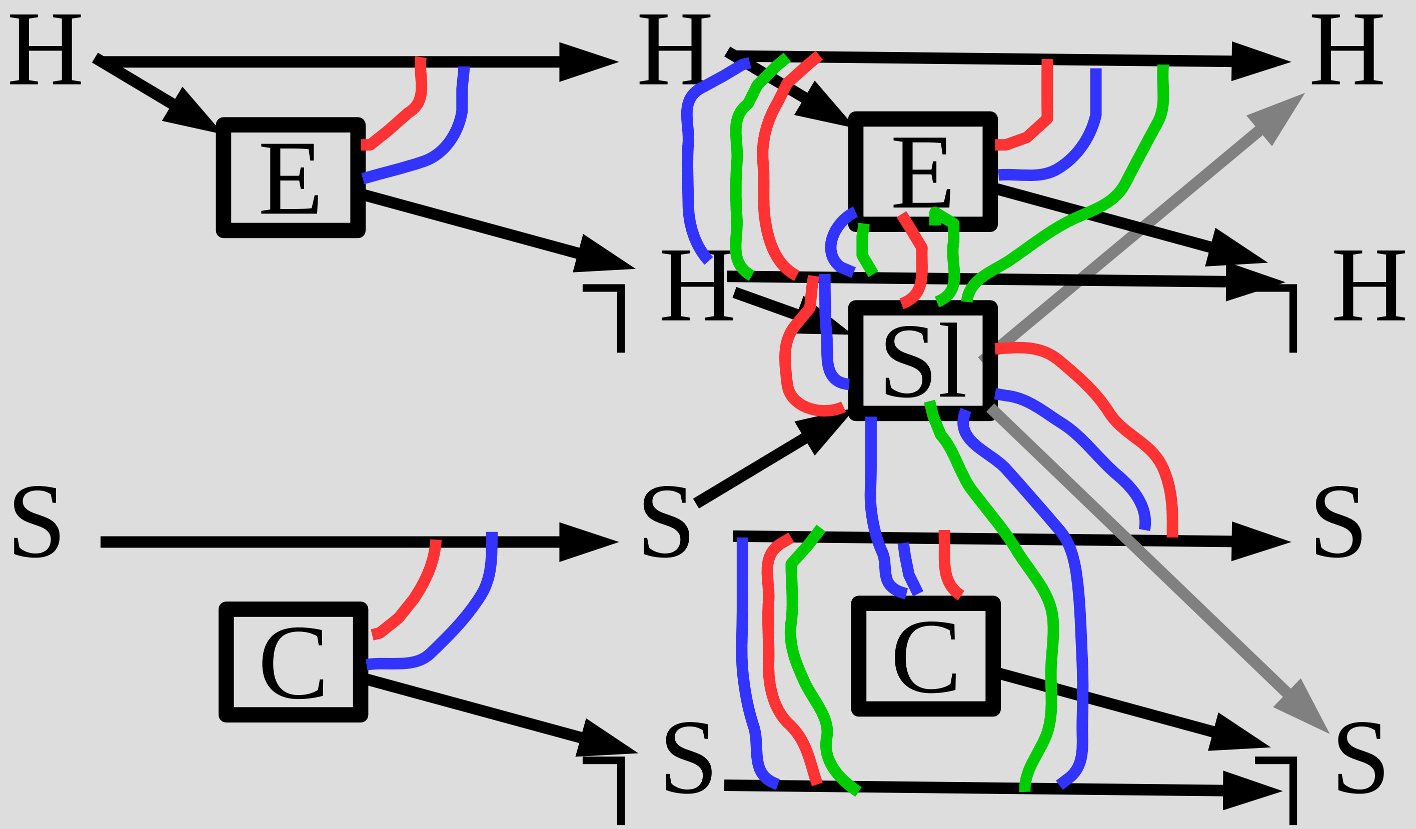
Action($Coffee(x)$,
Precondition: ,
Effect: $\neg Sleepy(x)$)

Action($Sleep(x)$,
Precondition: $Sleepy(x) \wedge \neg Hungr(x)$,
Effect: $\neg Sleepy(x) \wedge Hungry(x)$)

Mutexes: actions

Mutex Action rules:

1. $x \in Effect(A1) \wedge \neg x \in Effect(A2)$
2. $x \in Pre(A1) \wedge \neg x \in Effect(A2)$
3. $x \in Pre(A1) \wedge \neg x \in Pre(A2)$



Mutexes: states

There are 2 rules for states, but unlike action-mutexes they can change across levels

1. Opposite relations are mutexes (x and ¬ x)
2. If there are mutexes between all possible actions that “lead” to a pair of states...

Two ways that “leading” can be in mutex:
1. Actions are in mutex
2. Preconditions of action pair are in mutex

Mutexes: states

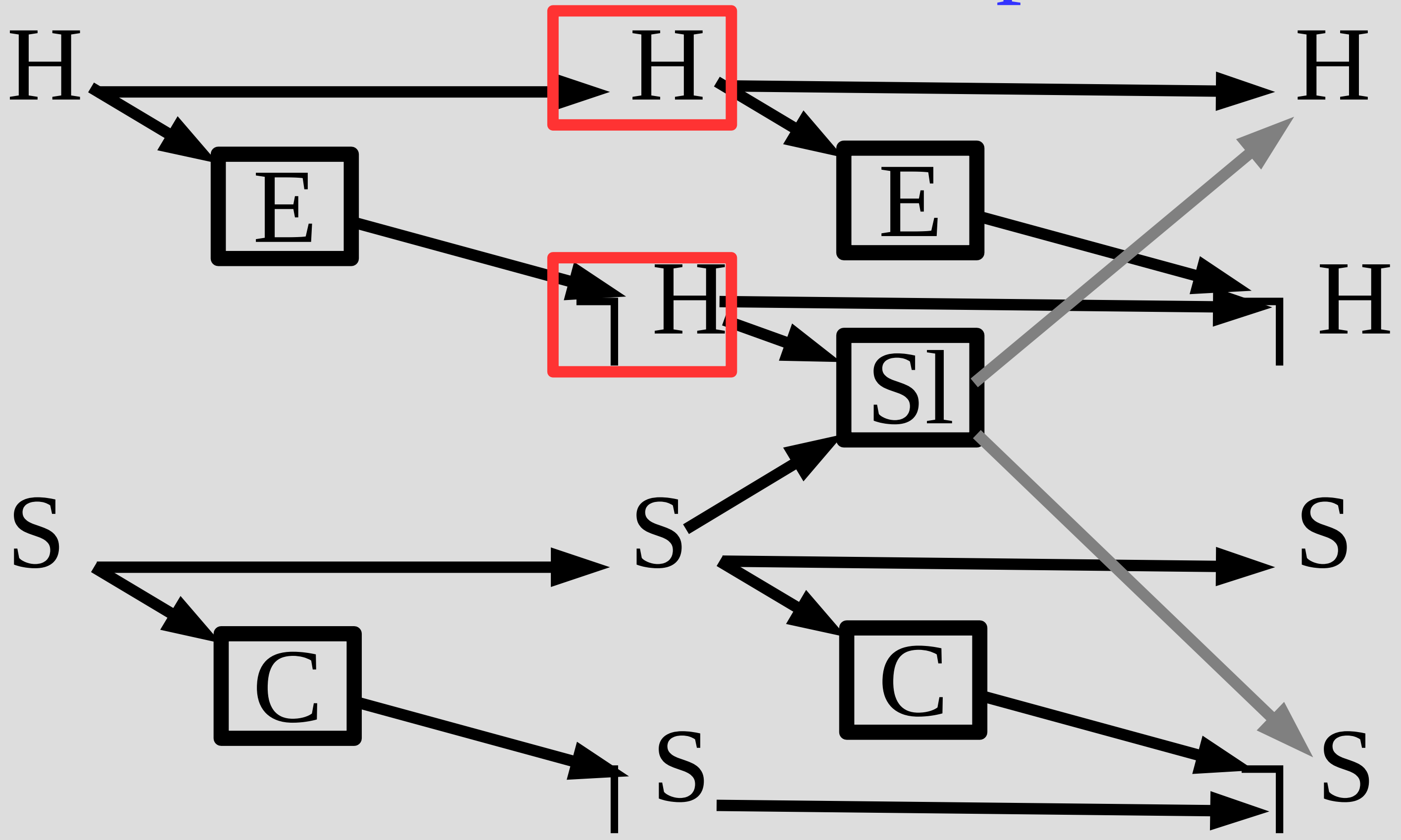
Another way to compute state mutexes:

(1) Add mutexes between all pairs in state
(2) If any pair of actions can lead to this pair of relationships, un-mutex them

Recap:
If any valid pair of actions = no mutex
All ways of reaching invalid = mutex

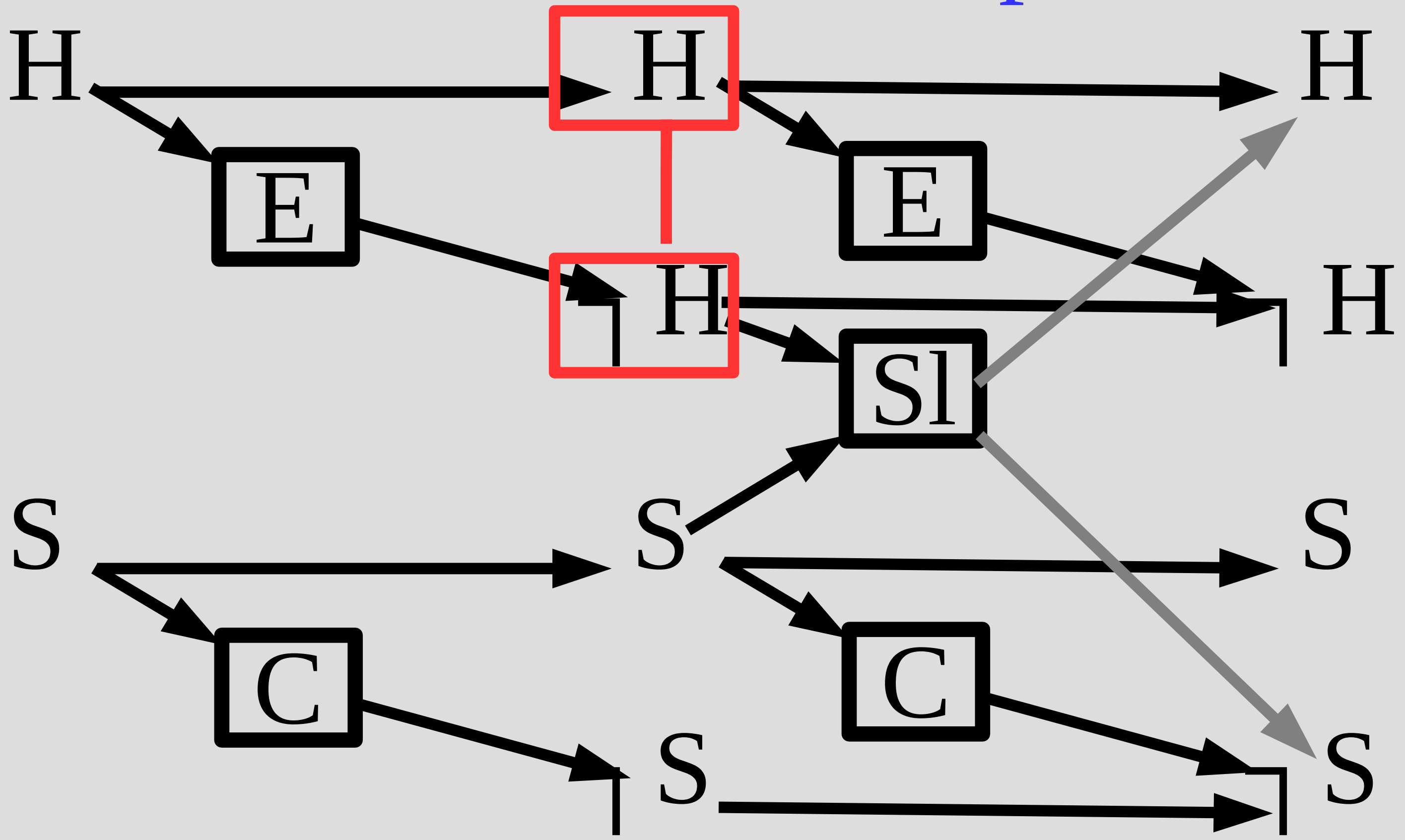
Mutexes: states

1. Opposite relations are mutexes (x and ┐x)
2. If there are mutexes between all possible actions that lead to a pair of states



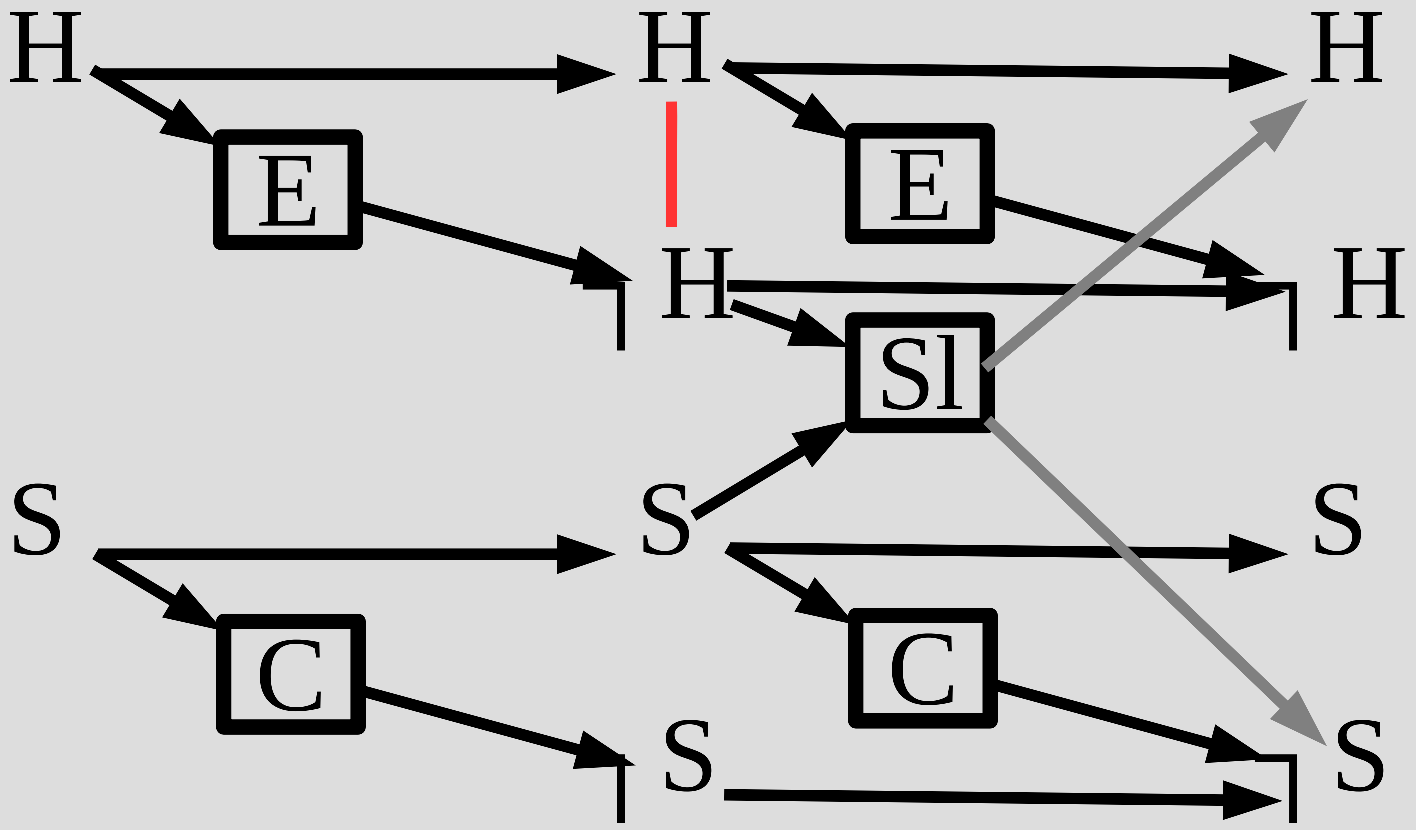
Mutexes: states

1. Opposite relations are mutexes (x and ┐x)
2. If there are mutexes between all possible actions that lead to a pair of states



Mutexes: states

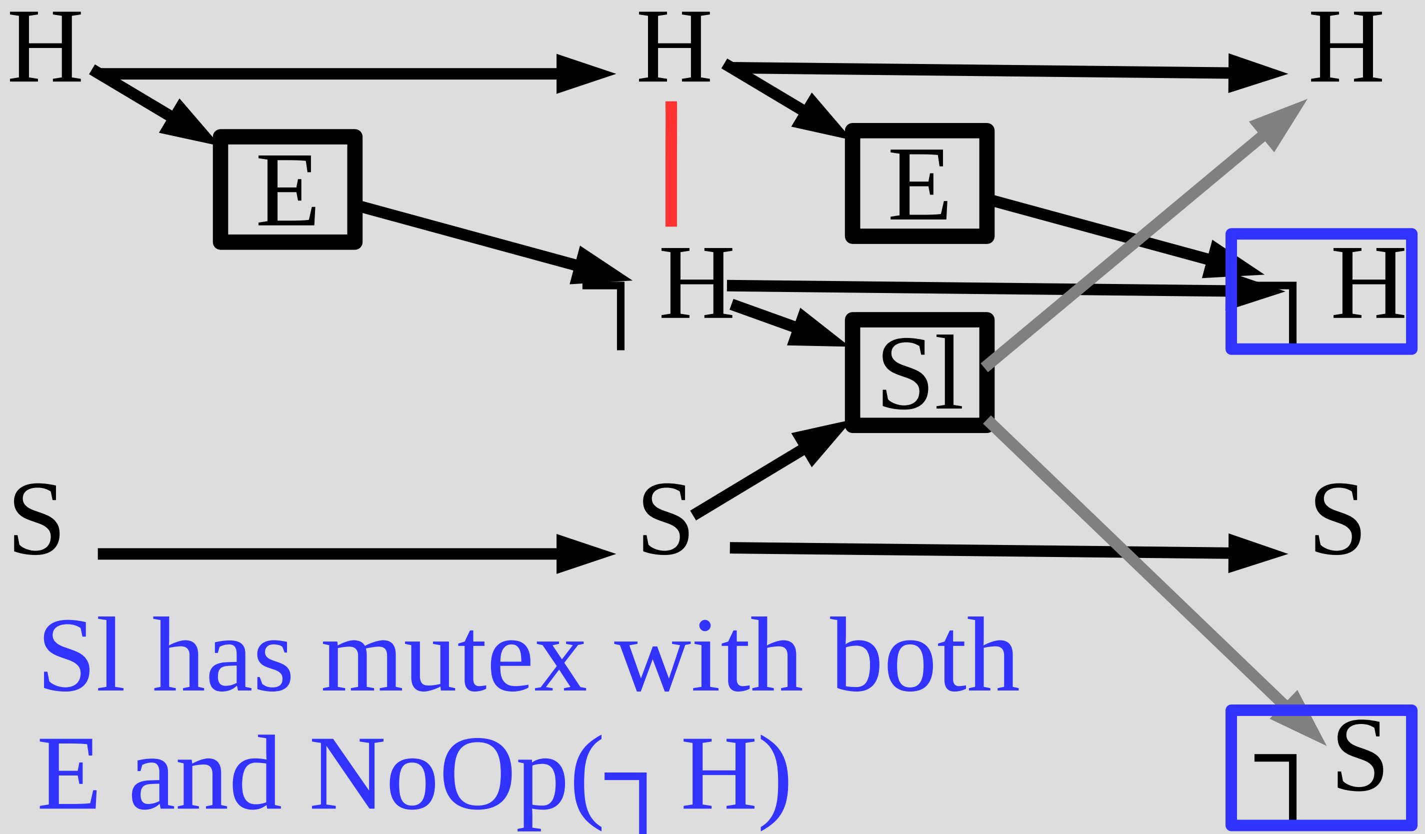
1. Opposite relations are mutexes (x and ¬ x)
2. If there are mutexes between all possible actions that lead to a pair of states



None... but if we remove coffee...

Mutexes: states

1. Opposite relations are mutexes (x and ┐ x)
2. If there are mutexes between all possible actions that lead to a pair of states

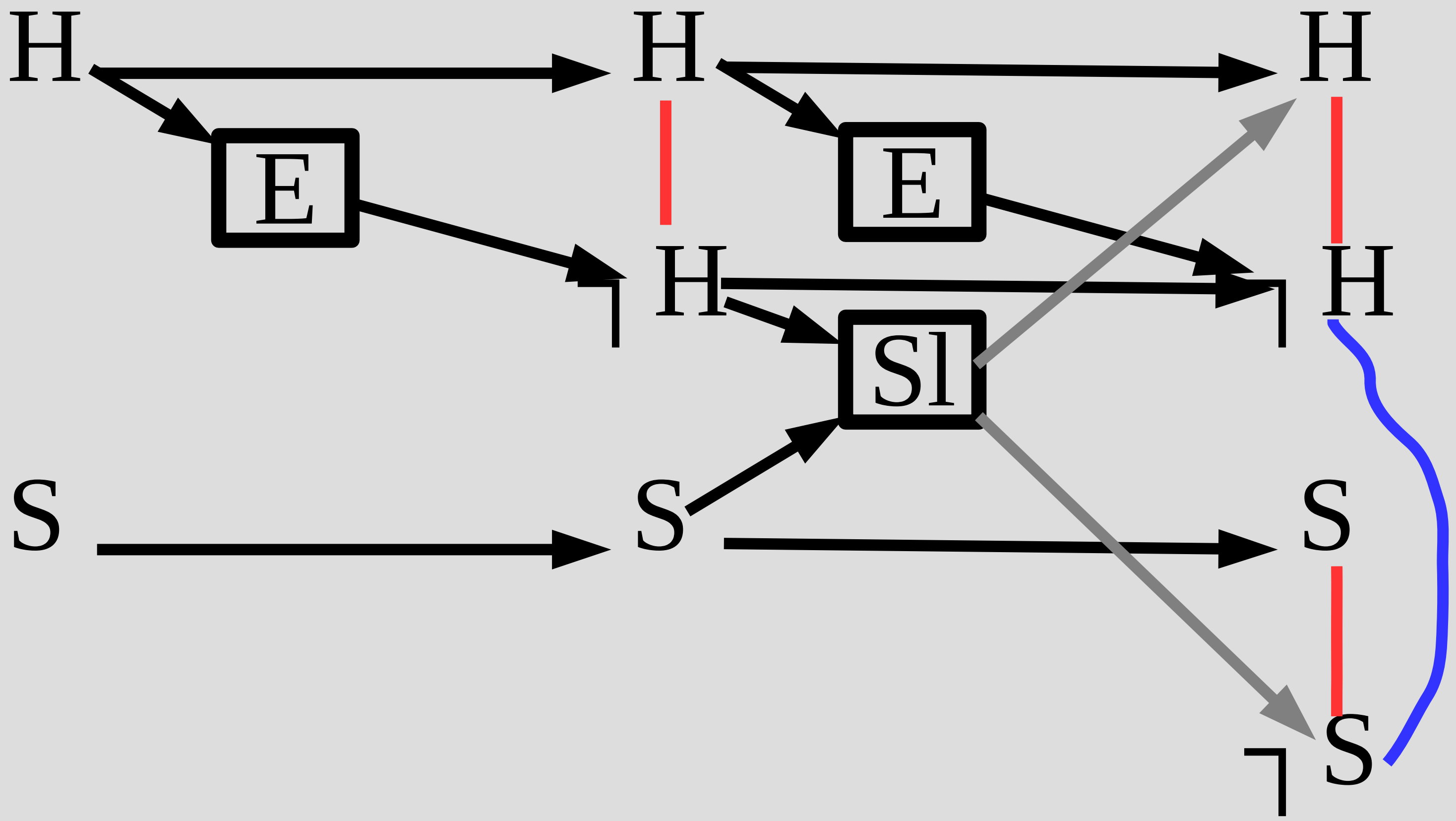


This mutex will be gone on the next level (as you can eat again)

Sl has mutex with both E and NoOp(┐ H)

Mutexes: states

1. Opposite relations are mutexes (x and ¬ x)
2. If there are mutexes between all possible actions that lead to a pair of states



Mutexes: actions

Consider...

Initial: $\neg Money \wedge \neg Smart \wedge \neg Debt$

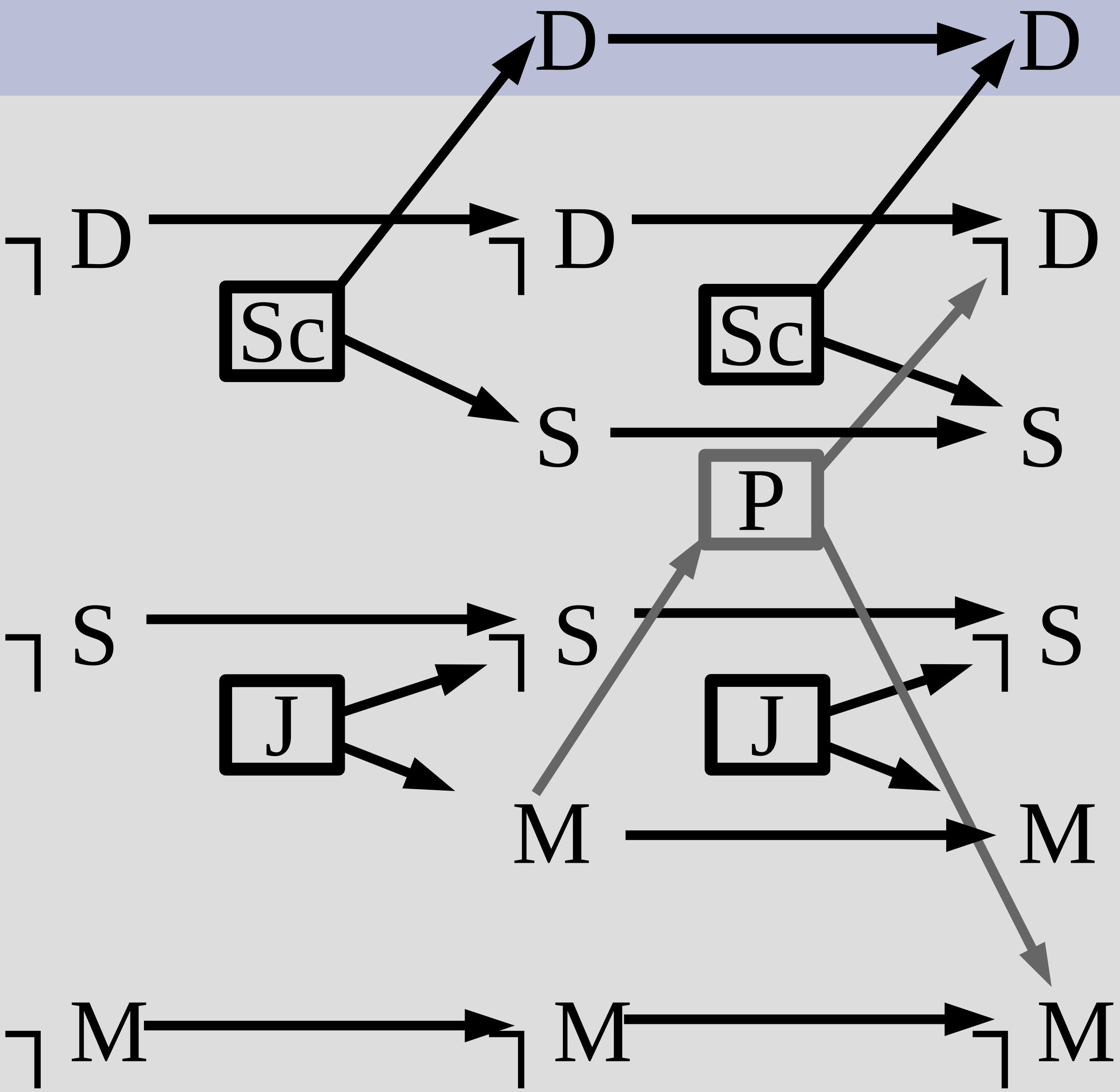
Goal: $\neg Money \wedge Smart \wedge \neg Debt$

Action($School$,
Precondition: ,
Effect: $Debt \wedge Smart$)

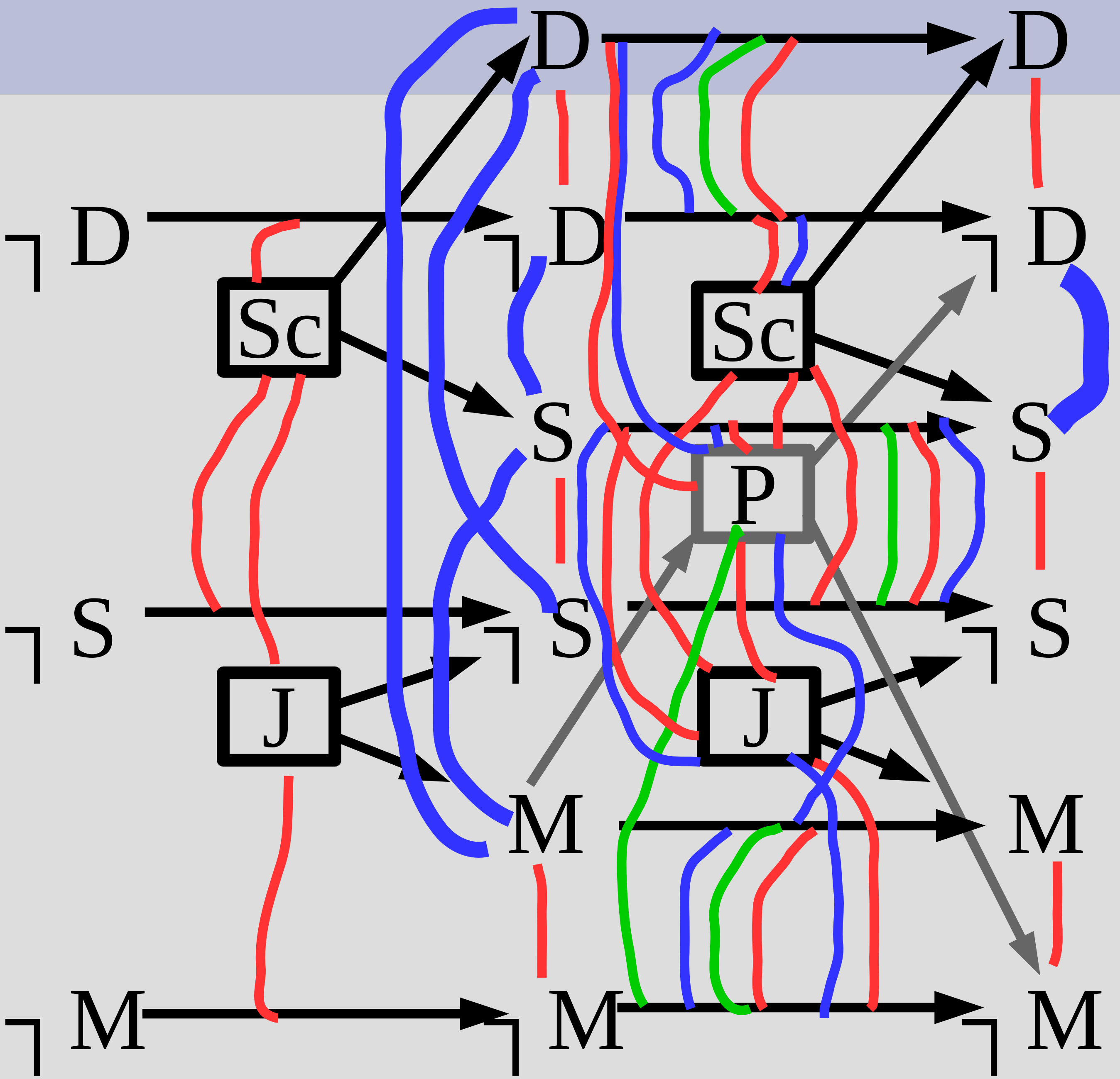
Action(Job,
Precondition: ,
Effect: $Money \wedge \neg Smart$)

Action(Pay,
Precondition: $Money$,
Effect: $\neg Money \wedge \neg Debt$)

Mutexes: actions



Mutexes: actions



Non-trivial mutexes:
(SC, P),
(J, P),
(SC, J),
(P,¬ D&M),
(SC,¬ D&¬ S),
(J,¬ M&S)

GraphPlan

GraphPlan can be computed in $O(n(a+l)^2)$,
where n = levels before convergence
a = number of actions
l = number of relations/literals/states
(square is due to needing to check all pairs)

The original planning problem is PSPACE, which is known to be harder than NP

GraphPlan: states

Let's consider this problem:

Initial: $Clean \wedge Garbage \wedge Quiet$

Goal: $Food \wedge \neg Garbage \wedge Present$

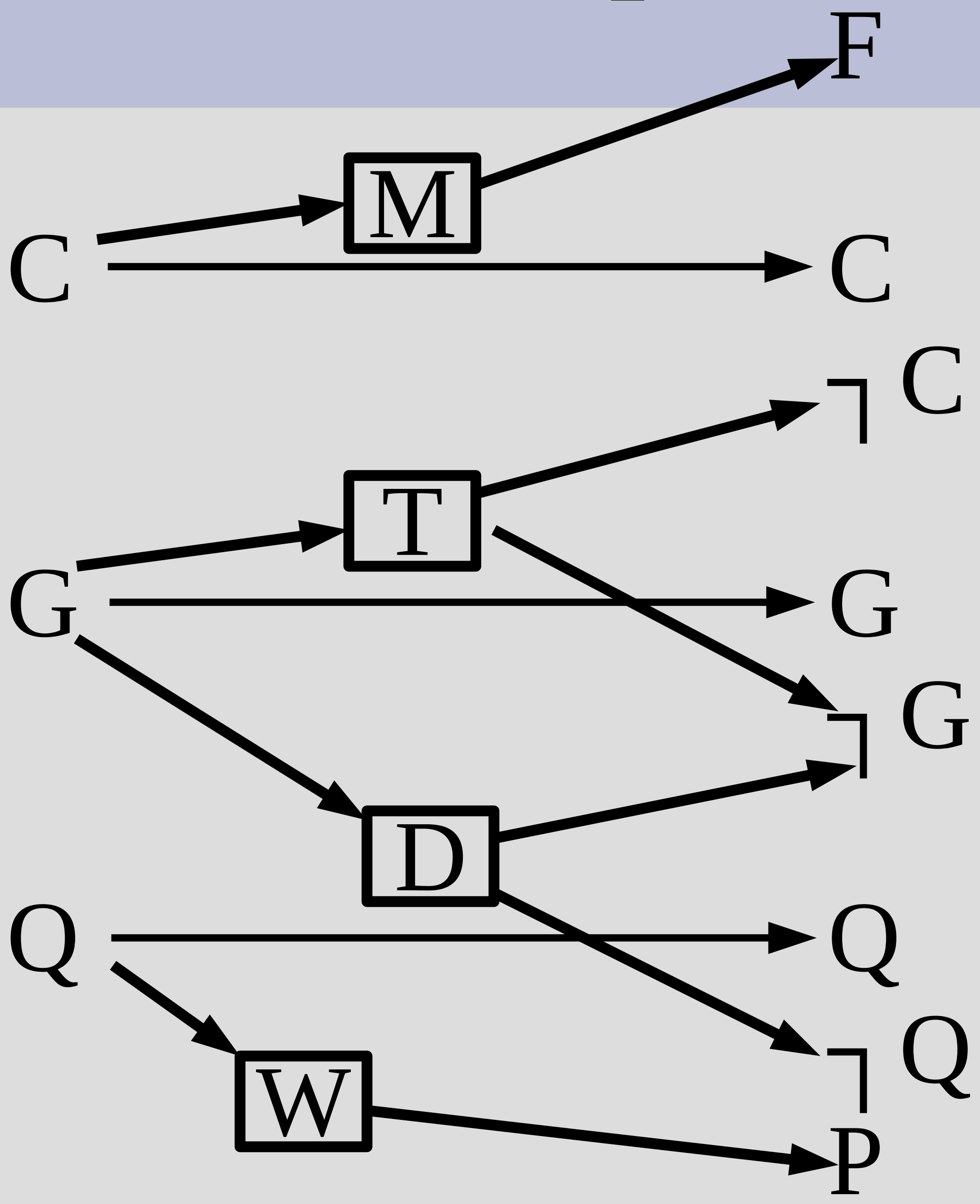
Action: ($MakeFood$,
Precondition: $Clean$,
Effects: $Food$)

Action: ($Takeout$,
Precondition: $Garbage$,
Effects: $\neg Garbage \wedge \neg Clean$)

Action: ($Wrap$,
Precondition: $Quiet$,
Effects: $Present$)

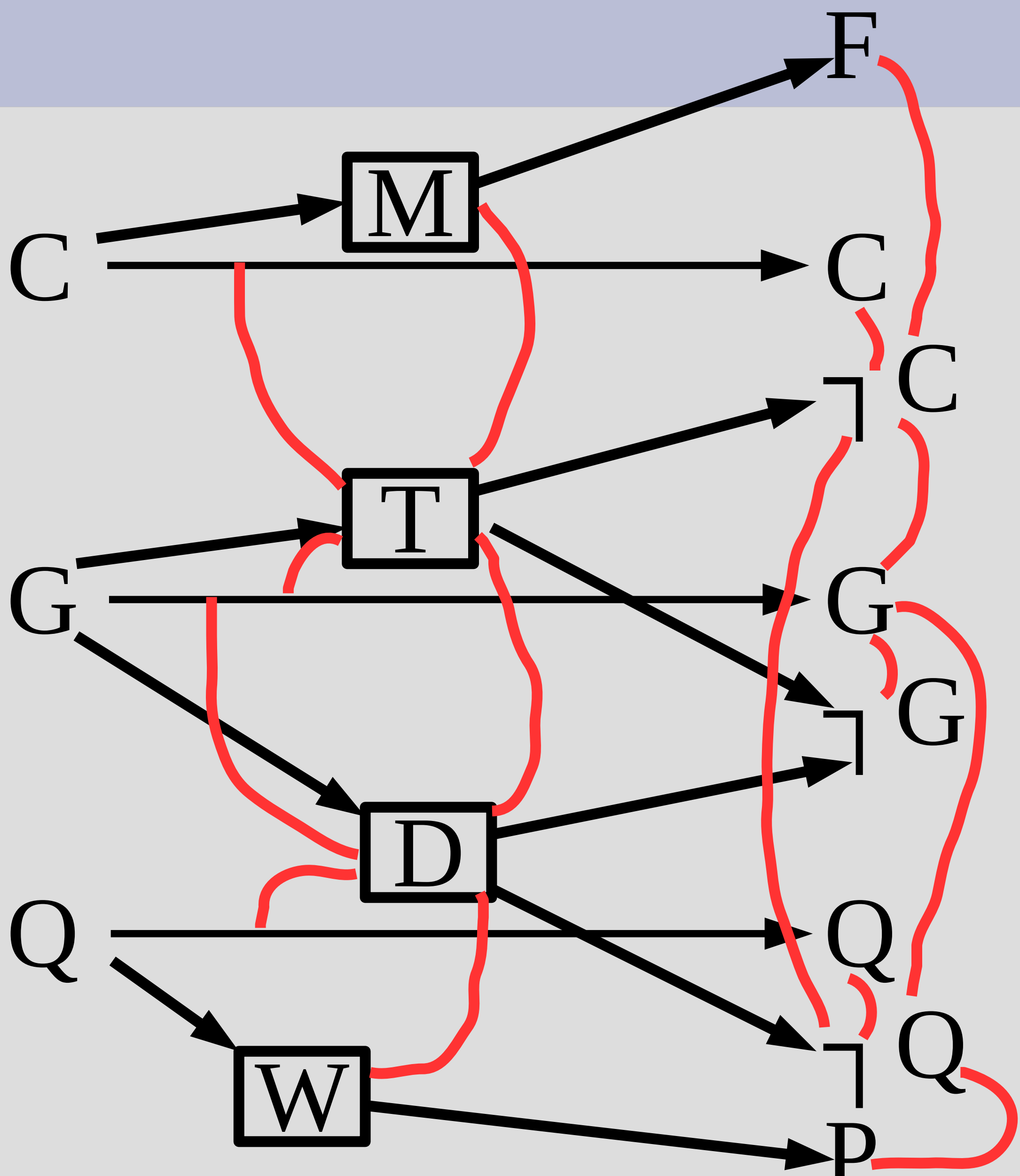
Action: ($Dolly$,
Precondition: $Garbage$,
Effects: $\neg Garbage \wedge \neg Quiet$)

GraphPlan: states



Take out one more level

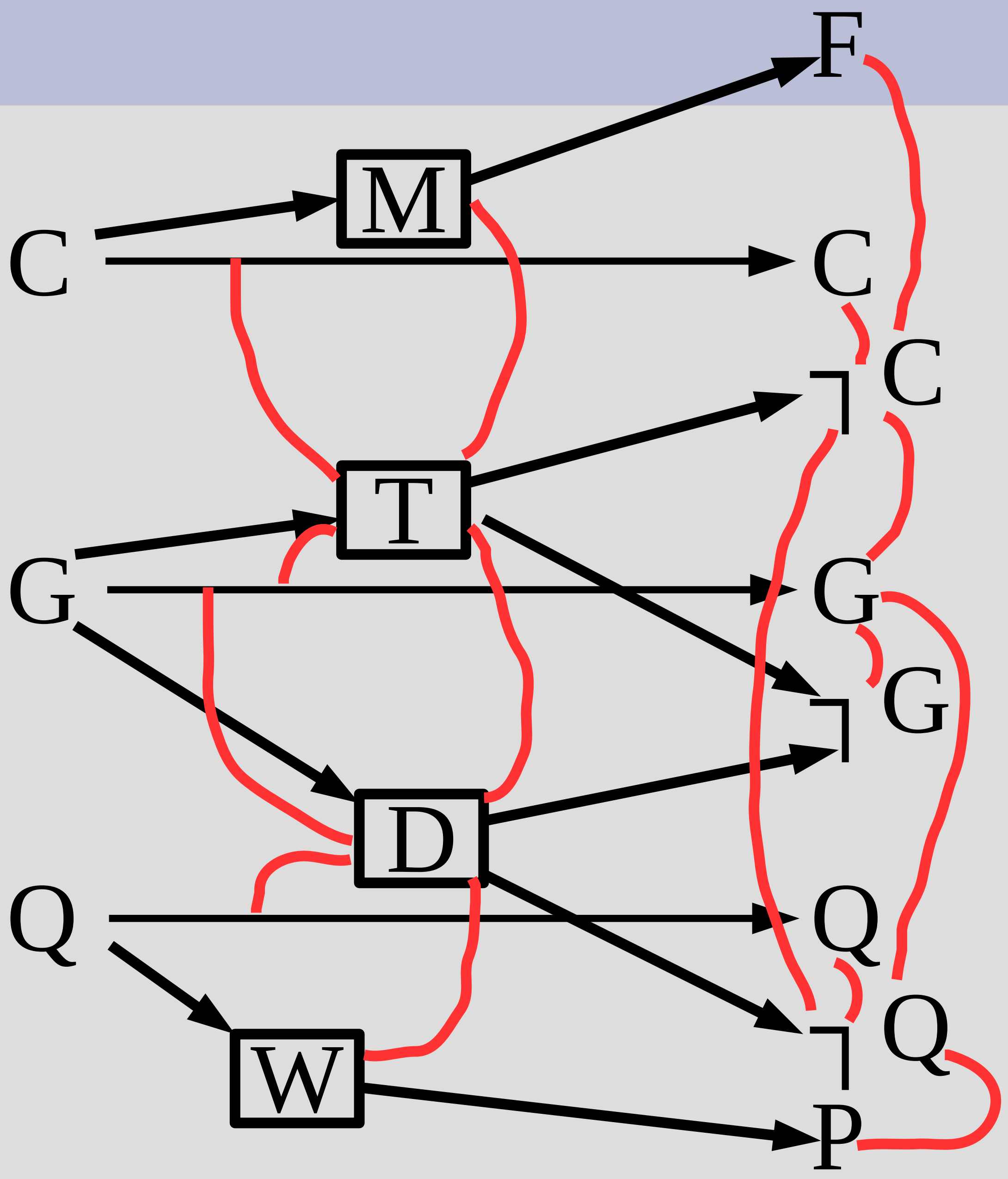
Mutexes



Possible state pairs:

F, C	C, Q
~~F, ┐C~~	C, ┐Q
F, G	C, P
F, ┐G	~~┐C, G~~
F, Q	┐C, ┐G
F, ┐Q	┐C, Q
F, P	~~┐C, ┐Q~~
~~C, ┐C~~	┐C, P
C, G	... (more)
C, ┐G	

Mutexes

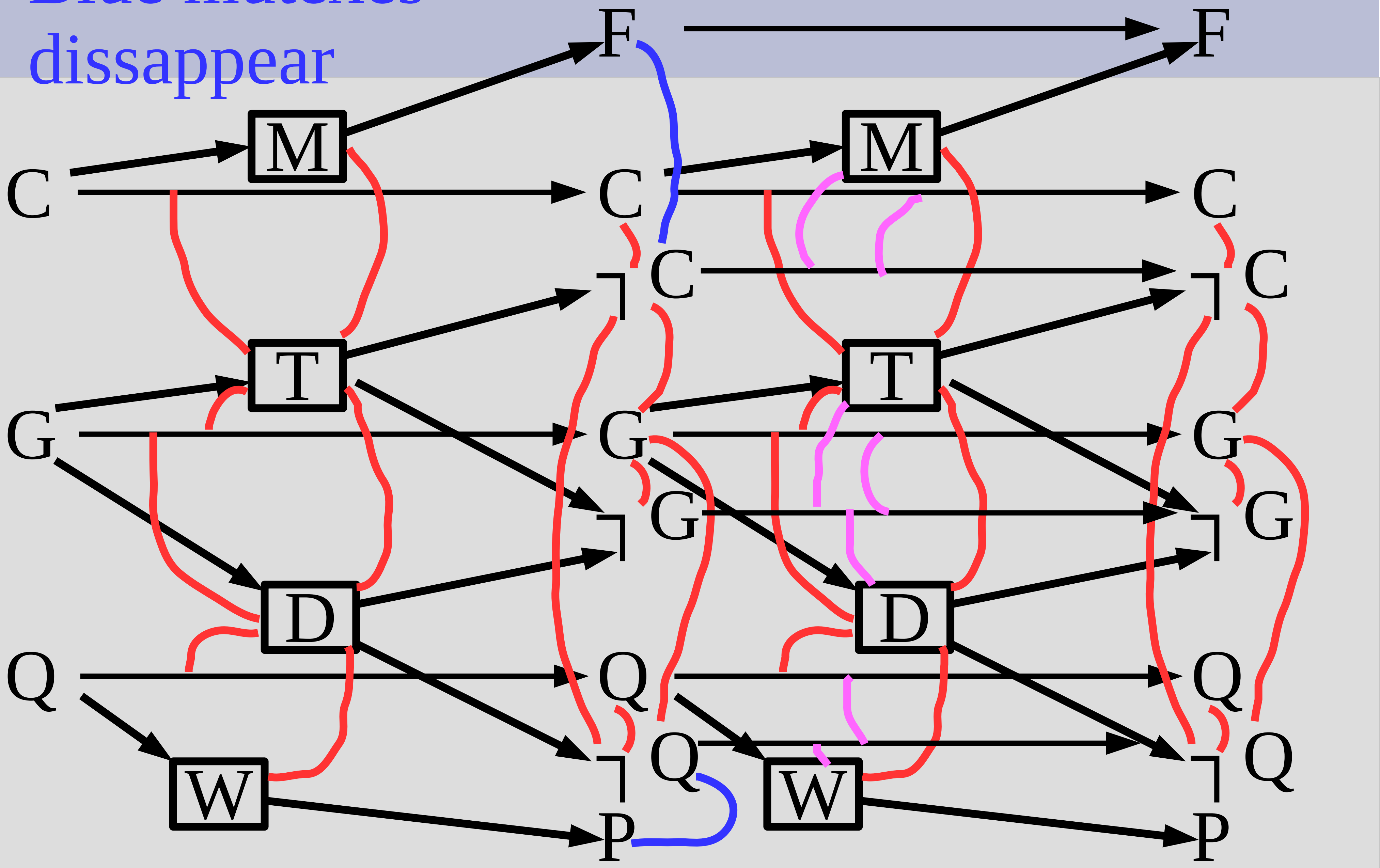


Make one more level here!

Mutexes

Blue mutexes dissappear

Pink = new mutex



GraphPlan as heuristic

GraphPlan is optimistic, so if any pair of goal states are in mutex, the goal is impossible

3 basic ways to use GraphPlan as heuristic:

(1) Maximum level of all goals
(2) Sum of level of all goals (not admissible)
(3) Level where no pair of goals is in mutex

(1) and (2) do not require any mutexes, but are less accurate (quick 'n' dirty)

GraphPlan as heuristic

For heuristics (1) and (2), we relax as such:

1. Multiple actions per step, so can only take fewer steps to reach same result
2. Never remove any states, so the number of possible states only increases

This is a valid simplification of the problem, but it is often too simplistic directly

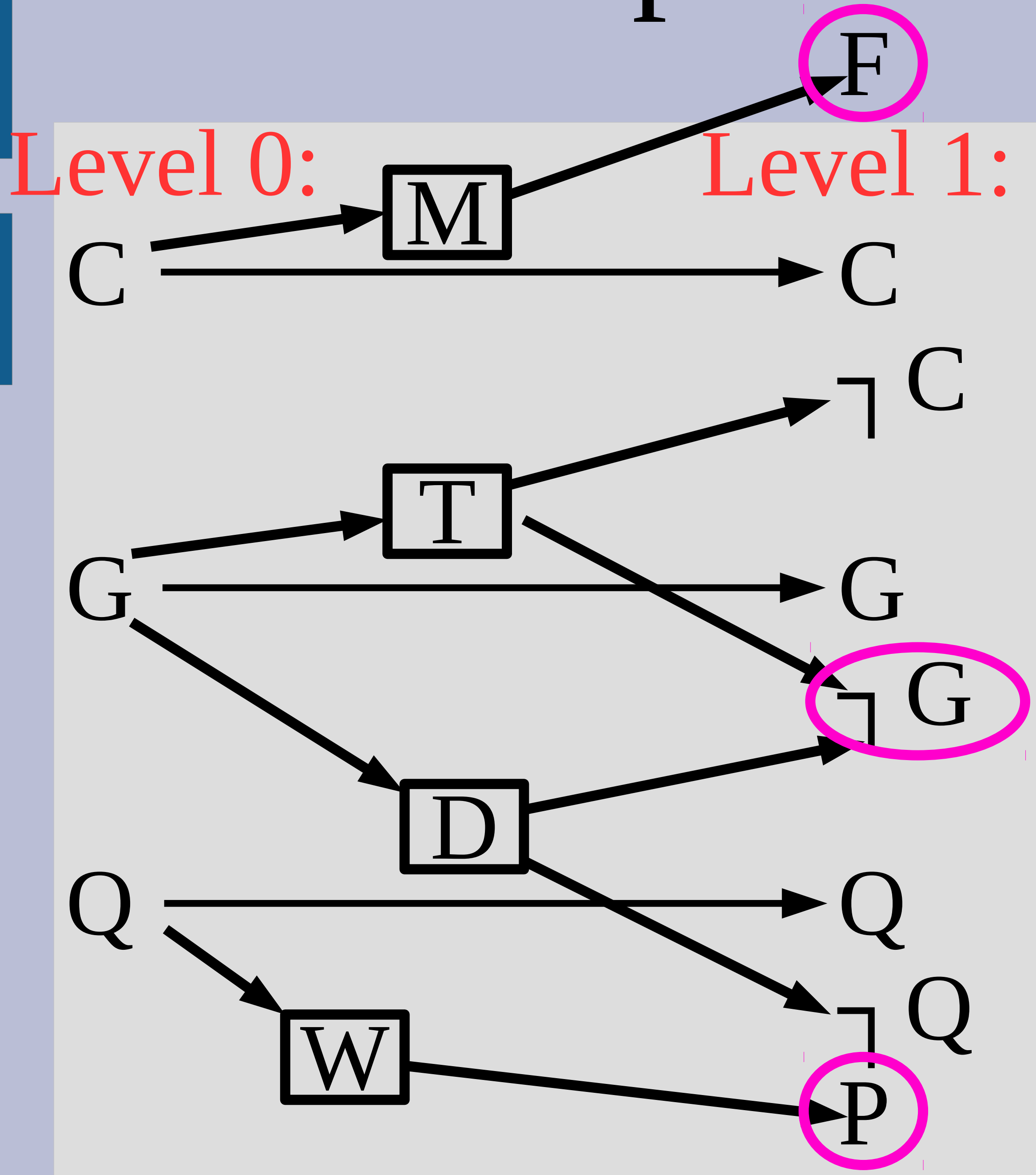
GraphPlan as heuristic

Heuristic (1) directly uses this relaxation and finds the first time when all 3 goals appear at a state level

(2) tries to sum the levels of each individual first appearance, which is not admissible (but works well if they are independent parts)

Our problem: goal={Food, ¬ Garbage, Present}
First appearance: F=1, ¬ G=1, P=1

GraphPlan: states



Heuristic (1):
Max(1,1,1) = 1

Heuristic (2):
1+1+1=3

GraphPlan as heuristic

Often the problem is too trivial with just those two simplifications

So we add in mutexes to keep track of invalid pairs of states/actions

This is still a simplification, as only impossible state/action pairs in the original problem are in mutex in the relaxation

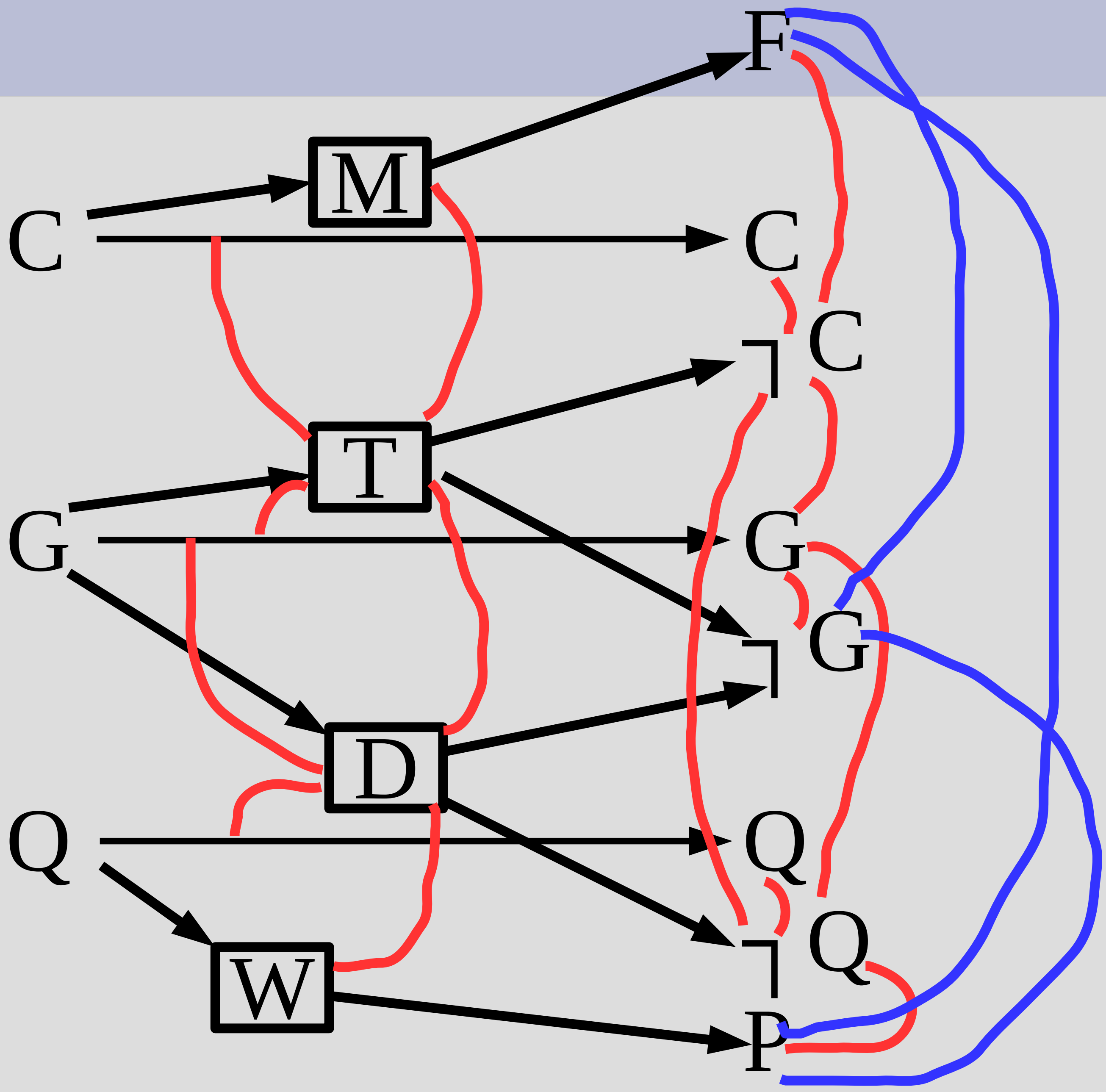
GraphPlan as heuristic

Heuristic (3) looks to find the first time none of the goal pairs are in mutex

For our problem, the goal states are:
(Food, ¬ Garbage, Present)

So all pairs that need to have no mutex:
(F, ¬ G), (F, P), (¬ G, P)

Mutexes



None of the pairs are in mutex at level 1

This is our heuristic estimate

Finding a solution

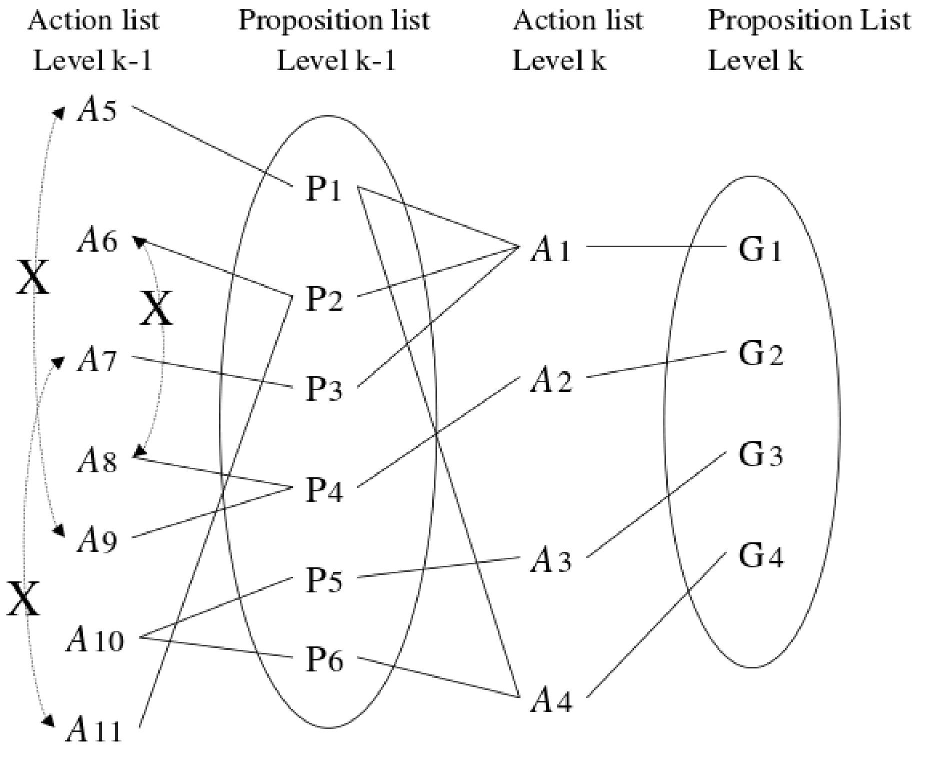
GraphPlan can also be used to find a solution:
(1) Converting to a Constraint Sat. Problem
(2) Backwards search

Both of these ways can be run once GraphPlan has all goal pairs not in mutex (or converges)

Additionally, you might need to extend it out a few more levels further to find a solution (as GraphPlan underestimates)

GraphPlan as CSP

Variables = states, Domains = actions out of
Constraints = mutexes & preconditions



(a) Planning Graph

Variables: $G_1, \cdots, G_4, P_1 \cdots P_6$

Domains: $G_1: \{A_1\}, G_2: \{A_2\} G_3: \{A_3\} G_4: \{A_4\}$
$P_1: \{A_5\} P_2: \{A_6, A_{11}\} P_3: \{A_7\} P_4: \{A_8, A_9\}$
$P_5: \{A_{10}\} P_6: \{A_{10}\}$

Constraints (normal): $P_1 = A_5 \Rightarrow P_4 \neq A_9$
$P_2 = A_6 \Rightarrow P_4 \neq A_8$
$P_2 = A_{11} \Rightarrow P_3 \neq A_7$

Constraints (Activity): $G_1 = A_1 \Rightarrow Active\{P_1, P_2, P_3\}$
$G_2 = A_2 \Rightarrow Active\{P_4\}$
$G_3 = A_3 \Rightarrow Active\{P_5\}$
$G_4 = A_4 \Rightarrow Active\{P_1, P_6\}$

Init State: $Active\{G_1, G_2, G_3, G_4\}$

(b) DCSP

from Do & Kambhampati

Finding a solution

For backward search, attempt to find arrows back to the initial state(without conflict/mutex)

Start by finding actions that satisfy all goal conditions, then recursively try to satisfy all of the selected actions’ preconditions

If this fails to find a solution, mark this level and all the goals not satisfied as: (level, goals) (level, goals) stops changing, no solution

Graph Plan

Remember this...

Initial: $\neg Money \wedge \neg Smart \wedge \neg Debt$

Goal: $\neg Money \wedge Smart \wedge \neg Debt$

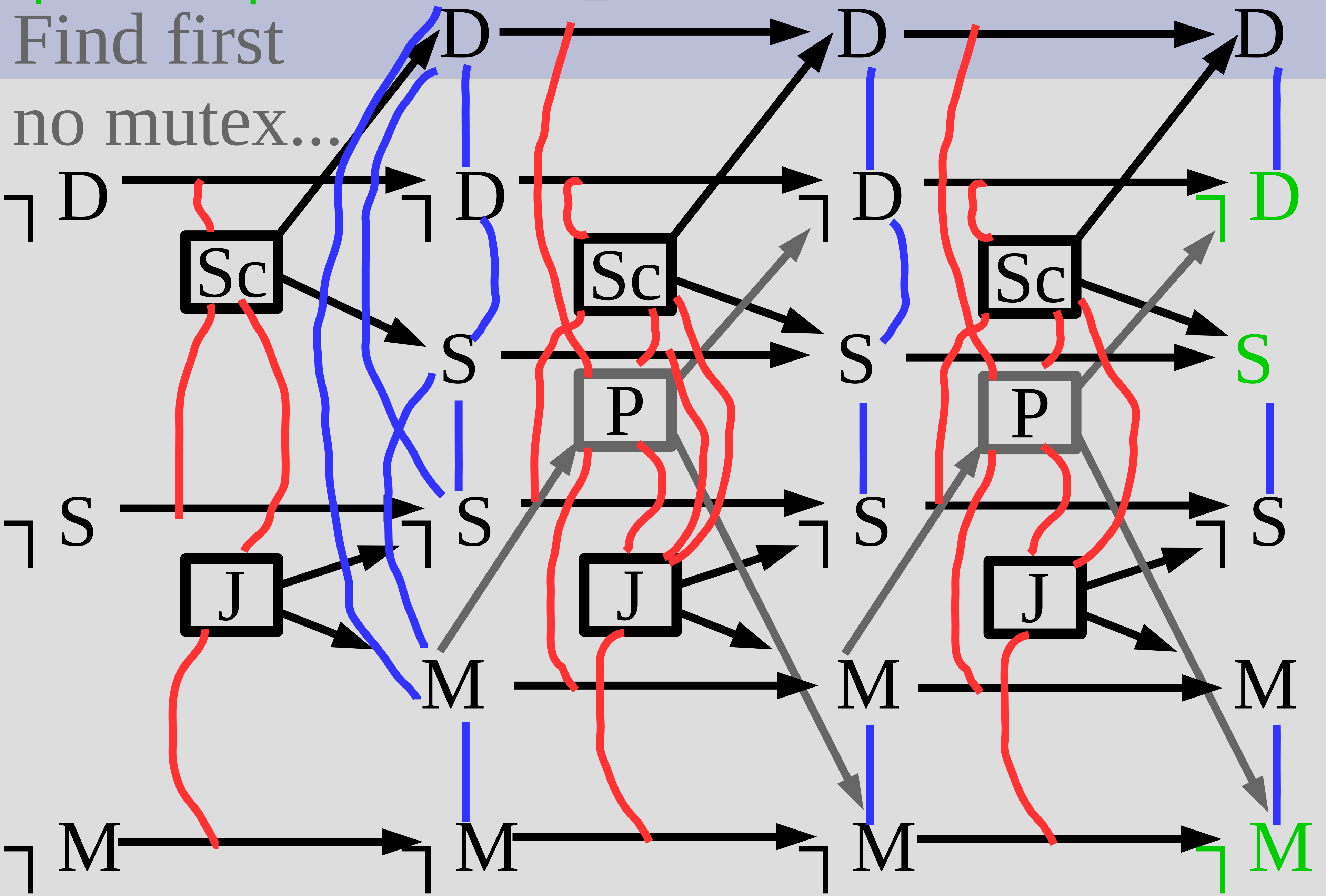
Action($School,$
Precondition: $,$
Effect: $Debt \wedge Smart)$

Action($Job,$
Precondition: $,$
Effect: $Money \wedge \neg Smart)$

Action($Pay,$
Precondition: $Money,$
Effect: $\neg Money \wedge \neg Debt)$

Graph Plan

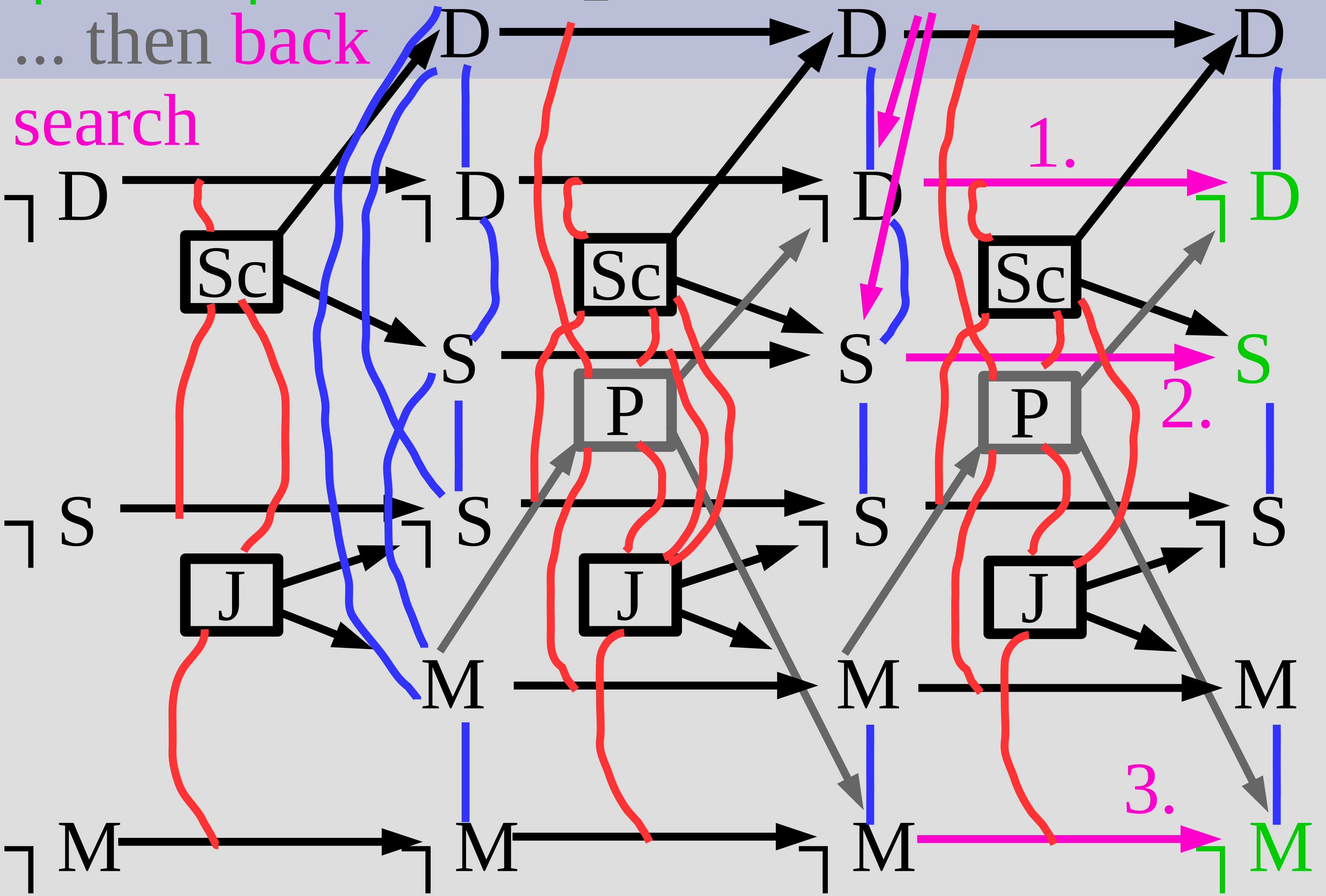
Ask:
¬D^S^¬M
Find first
no mutex...



Graph Plan

Ask:
¬D^S^¬M
... then back search

Error! States of 1&4 in mutex



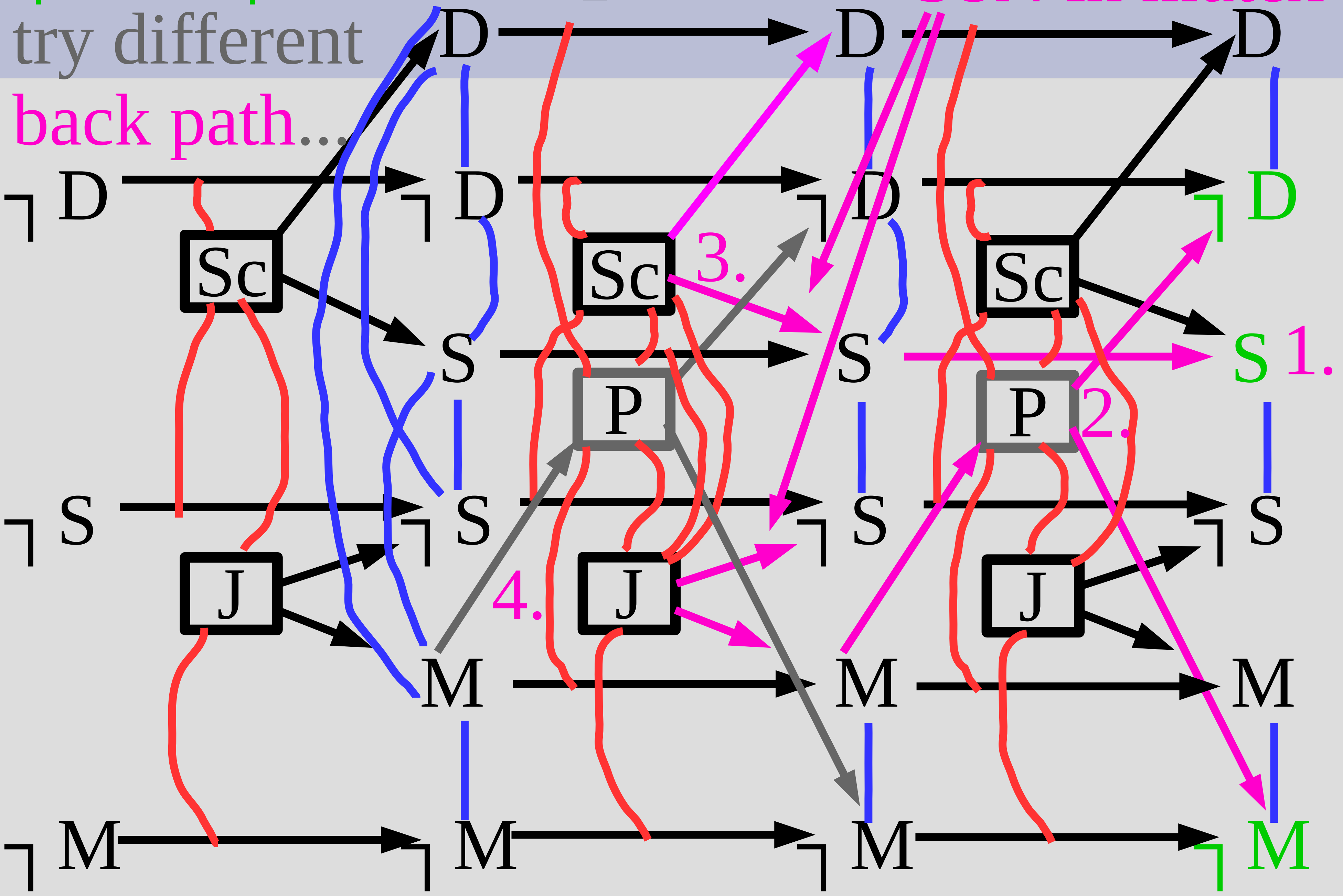
Graph Plan

Ask:

¬D^S^¬M

try different back path...

Error, actions 3&4 in mutex



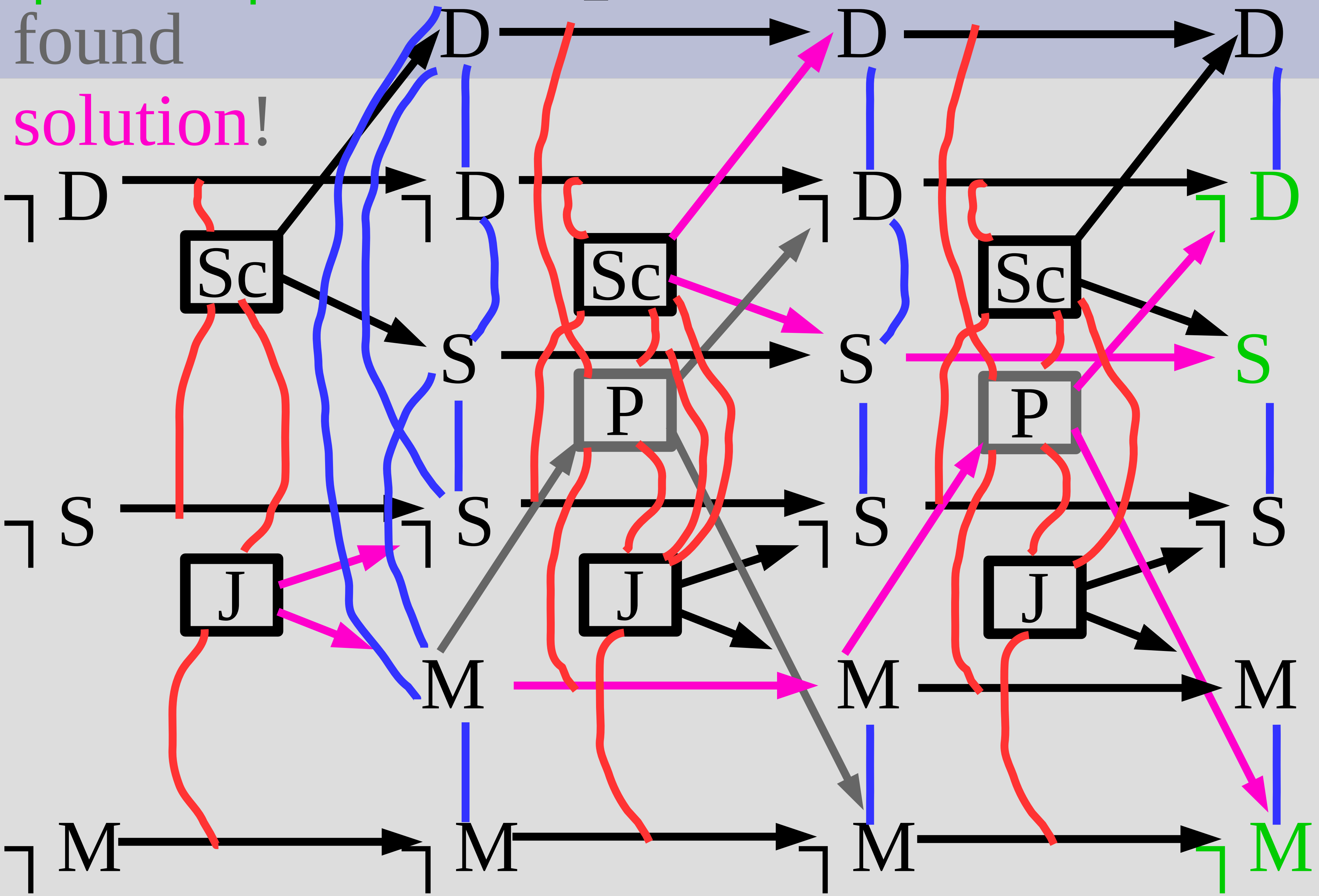
Graph Plan

Ask:

┐D^S^┐ M

found

solution!



Finding a solution

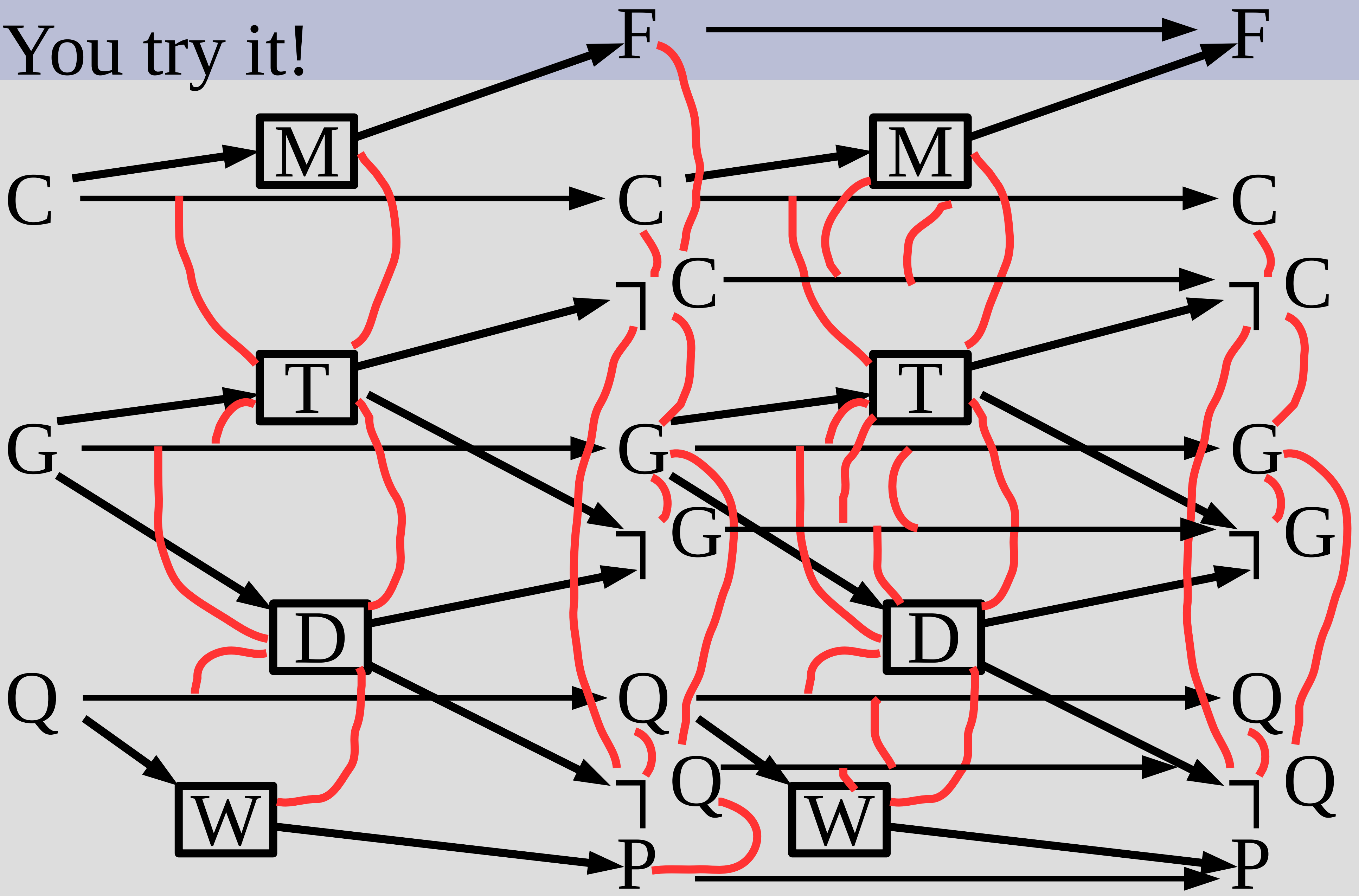
Formally, the algorithm is:

```
graph = initial
noGoods = empty table (hash)
for level = 0 to infinity
    if all goal pairs not in mutex
        solution = recursive search with noGoods
        if success, return paths
    if graph & noGoods converged, return fail
    graaph = expand graph
```

Initial: $Clean \wedge Garbage \wedge Quiet$

Goal: $Food \wedge \neg Garbage \wedge Present$

You try it!


F
F
M
M
C
C
C
¬C
¬C
T
T
G
G
G
¬G
¬G
D
D
Q
Q
Q
¬Q
¬Q
W
W
¬P
¬P
P
P