Massachusetts Institute of Technology

6.034 Artificial Intelligence

Examination Solutions # **1**

Problem 1

Part A (5 points)

The rule for uncle is:

```
IF ((? grandparent) is-a-parent-of (? parent))
   ((? parent) is-a-parent-of (? person))
   ((? grandparent) is-a-parent-of (? uncle))
   ((? uncle) is-of-gender Male)
AND-IF (not (eq (? uncle) (? parent)))
THEN ((? uncle) is-uncle-of (? person))
```

Part B (4 points)

The primary source of the slowdown is the JOIN operation in Rete's beta nodes. This operation is there to enforce consistent variable bindings among the IF conditions of the rules. Note that any newly added `is-a-parent-of` assertion produces new bindings at three alpha nodes. Each of these bindings have to be checked (in the beta nodes) against all the bindings in some other alpha node. That is, the JOIN operation is $O(N^2)$ where N is the number of entries in the alpha nodes, which in this case is millions of `is-a-parent-of` entries.

Part C (3 points)

To find your maternal grandmother's uncles you should use `(backchain '((? x) is-uncle-of Granny))`. This will constrain you to look at relevant assertions, those involving `Granny`. In forward chaining you would end up generating every possible uncle in the database and then you would have to find the appropriate one. Some people were under the impression that backward chaining could only find one answer, which is not true (for example, note the `*find-all*` switch in PS3).

Part D (5 points)

The robot would go into an infinite loop among rooms R1, R2, R5, R6.

Step	Room the robot is in: (loc .))
1.	R1
2.	R2
3.	R5
4.	R6
5.	R1
6.	R2

This happens because the GO rule is always triggered and chosen to fire, since it is first, and the `(door r2 r5)` assertion is found before the `(door r2 r3)` assertion. So the robot never gets a chance to go to the other rooms.

Part E (4 points)

Adding the SYM rule would not change anything as long as it is placed at the end of the rules, the GO rule would fire first.

Part F (4 points)

In the general case, most of the conflict resolution strategies would not guarantee stopping at the target room. Of the options we gave you, only using "rule recency" as the primary method with "data recency" as the secondary (option 5) guarantees the desired result. We gave partial credit for the other combination of data recency and rule recency (option 4) as well. The key insight is that you have to give all the rules as well as all the assertions a chance to be involved in the firing so that all the possibilities are explored.

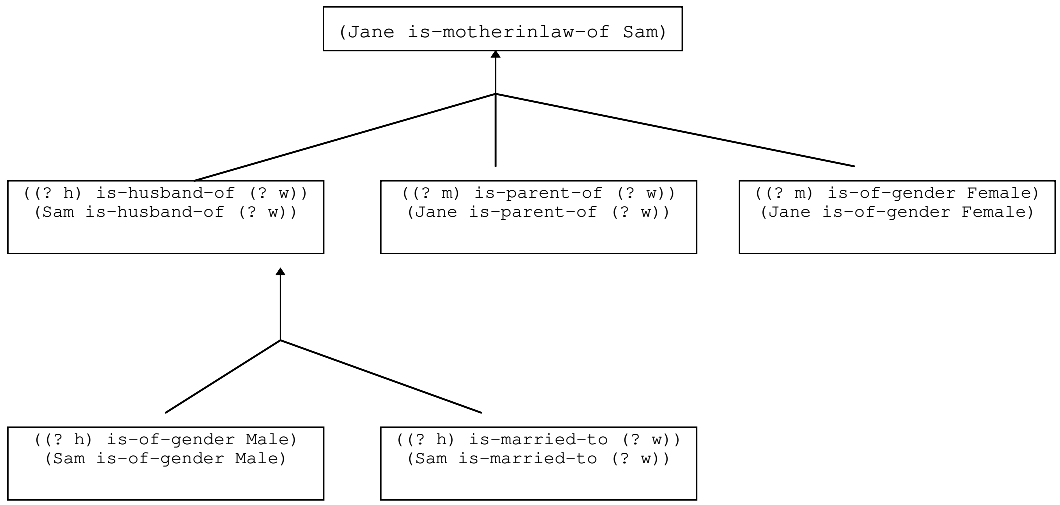
Option 1 (rule order) clearly does not work, as the rest of the problem showed. Option 6 (fire them all) is non-sensical when you have rules that do deletions. Option 2 (rule recency) is not enough because it does not address the problem that the `(door r2 r5)` assertion is found before the `(door r2 r3)` assertion. Option 3 (data recency) is not enough because we have to let the STOP rule have a chance to fire if we want the robot to halt. Option 4 (data recency and then rule recency) has a similar problem of not giving the STOP rule enough of a chance to fire.

Problem 2

Part A

The tree is shown below. The important part was to get the topology correct, and to produce a subgoal tree (as the question asked).

The sequence of database queries and questions is shown. Note that the standard backward chainer always queries the database, then tries rules, and then tries asking. You lost a small number of points for not indicating that the system checked the database for each goal.


(Jane is-motherinlaw-of Sam)
((? h) is-husband-of (? w))
(Sam is-husband-of (? w))
((? m) is-parent-of (? w))
(Jane is-parent-of (? w))
((? m) is-of-gender Female)
(Jane is-of-gender Female)
((? h) is-of-gender Male)
(Sam is-of-gender Male)
((? h) is-married-to (? w))
(Sam is-married-to (? w))

```
Database Queries                      Questions Asked
1) (Jane is-motherinlaw-of Sam)
   [not in db, use rule R1]
2) (Sam husband-of (?  w))
   [not in db, use rule R2]
3) (Sam is-of-gender Male)
   [not in db, no rules, ask]         4) Is it true that (Sam is-of-gender-Male)?  Y
5) (Sam is-married-to (?  w)
   [in db, w=Betty]
6) (Jane parent-of Betty)
   [in db]
7) (Jane is-of-gender-Female)
   [not in db, no rule, ask]          8) Is is true that (Jane is-of-gender Female)?  Y
```

Part B

The important idea is that the system can answer `Why` questions by displaying the goal one level up in the subgoal tree. The initial query is about whether `(Jane is-motherinlaw-of Sam)`, and the first subgoal is `(Sam is-husband-of (?  w))`. Hence if we asked why about that, the system would respond with something like

`I am trying to determine whether (Jane is-motherinlaw-of Sam).`

If we wanted to be especially helpful the system could go on to say

```
To do this I am using rule R1:
        if (Sam is-husband-of (? w)) and
           (Jane is-parent-of (? w)) and
           (Jane is-of-gender Female)
        then
           (Jane is-motherinlaw-of Sam)
```

Part C

There are a few valid answers here. One obvious one is

```
(IF
     ((? p) is-parent-of (? w))
     ((? h) is-husband-of (? w))
THEN ((? h) is-soninlaw-of (? p))
```

Note that you don't need to specify that (? h) is male; this is ensured by rule R2. Note also that (? p) should not be limited to being female (fathers can have son-in-laws also); but you didn't lose any points for this, as the question implied that we were interested in females having son-in-laws.

You can also use the is-motherinlaw-of rule from the previous question, but you need to be careful:

```
(IF
     ((? m) is-motherinlaw-of (? s))
     ((? s) is-of-gender Male)
THEN ((? s) is-soninlaw-of (? m))
```

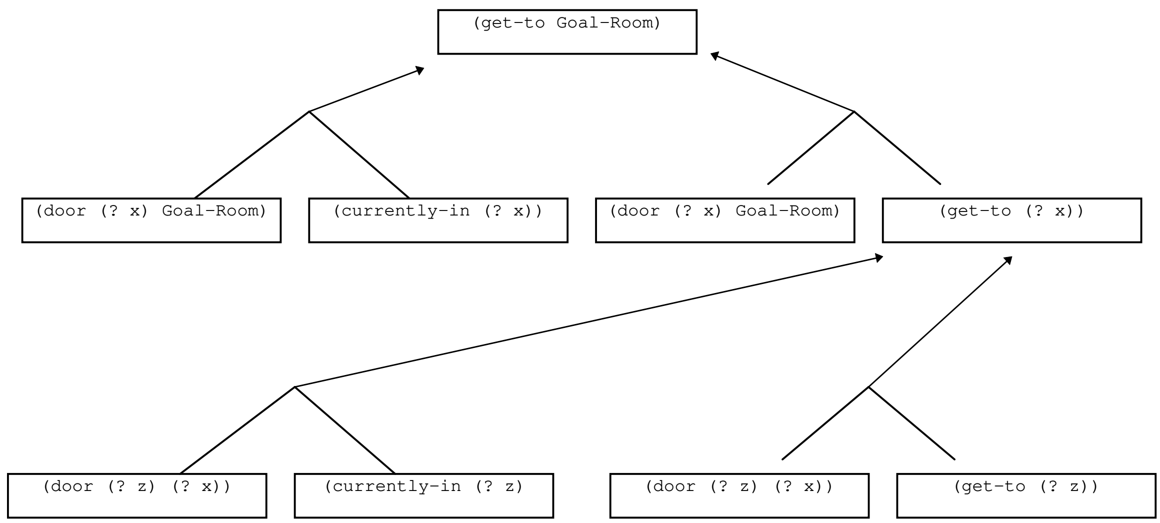
It is not enough to use the first clause alone; you do need the gender clause here or you wind up with females being called son-in-laws.

Part D

The tree is shown. Again here the important part was to get the topology correct, and to produce a subgoal tree (as the question asked).

You needed to realize that the `get-to` clause in Rule2 sprouts a second subtree.

The sequence of database queries and questions is shown below. You did not need to get every detail shown here, but you did need to indicate that the system searched through the rooms in the order shown. [The limit of 8 questions was simply to let you know you didn't have to shown the entire search tree (which is fairly large).]



(get-to Goal-Room)
(door (? x) Goal-Room)
(currently-in (? x))
(door (? x) Goal-Room)
(get-to (? x))
(door (? z) (? x))
(currently-in (? z)
(door (? z) (? x))
(get-to (? z))

Database Queries	**Questions Asked**
1) (get-to Goal-Room)	
[not in db, use rule R1]	
2) (door (? x) Goal-Room)	
[db yields x=R1]	
3) (curently-in R1)	
[not in db, no rules, ask]	4) Is it true that (currently-in R1)? N
5) [db yields x=R2]	
6) (curently-in R2)	
[not in db, no rules, ask]	7) Is it true that (currently-in R2)? N
8) [db yields x=R3]	
9) (curently-in R3)	
[not in db, no rules, ask]	10) Is it true that (currently-in R3)? N
11) [db yields x=R4]	
12) (curently-in R4)	
[not in db, no rules, ask]	13) Is it true that (currently-in R4)? N
13) [Rule1 fails on top level goal, try Rule2]	
14) (door (? x) Goal-Room)	
[db yields x=R1]	
15) (get-to R1)	
[not in db, try Rule2]	
16) (door (? z) R1)	
[change variable name to avoid conflict, db yields z=R5]	
17) (currently-in R5)	
[not in db, no rule, ask]	18) Is is true that (currently-in R5)? N
19) [db yields z=R6]	
20) (currently-in R6)	
[not in db, no rule, ask]	21) Is is true that (currently-in R6)? N
22) [Rule1 fails on (get-to R1), try Rule2]	
23) (door (? z) R1)	
[db yields z=R5]	
24) (get-to R5)	
[not in db, true Rule1]	
25) (door (? x) R5)	
[not in db, no rule, ask]	26) Is is true that (door (? x) R5)? N [an awkward way of asking, is there a door into R5]
26) [Rule1 failed, try Rule2 on subgoal 24]	
27) (door (? x) R5)	
[not in db, no rule, ask]	28) Is is true that (door (? x) R5)? N

Part E

Backward chaining systems are not appropriate where there is significant "fan-in,", i.e., many different paths leading away from the goal. Forward chaining works much better in these cases.

Problem 3

Part A (4 points)

You ask several friends for advice in finding a route. One suggests hill-climbing, **without** backtracking. That is, once a choice is made at a node there is no possibility of backing up and trying other choices.

The suggested method does not find a path to the goal. The search goes down through node 1 to the bottom of the circle of nodes, and without backup, the search terminates with no solution.

Including, of course, the start node, you have

$1 + (b \times d) = 1 + (2 \times 5) = 11$ nodes in the search tree.

Many, for some reason, included only the nodes on the path from S to the outer ring, ignoring the losing nodes in the binary expansions, and lost one point.

Part B (3 points)

Another friend suggests hill-climbing, **with** backtracking.

With backtracking hill climbing succees, because it eventually tries all paths. The search is misled by the node arrangement into trying the lower left quadrant, then the upper left, the lower right, and finally the upper right. Thus, all the nodes end up in the search tree.

Including the goal node in the start tree, you have

$1 + \frac{b^{d+1}-1}{b-1} = 1 + \frac{2^6-1}{2-1} = 64$ nodes in the search tree.

Of course, if you forgot and couldn't derive the formula, you could count.

Part C (3 points)

Another friend suggests breadth-first search.

Success, with all 64 nodes in the search tree, as breadth-first search expands paths level by leve.

If you have a number different from 64 for all the nodes in the tree, you lost points only once if you were consistent.

Part D (3 points)

Not yet having run out of friends, still another suggests that you use hill-climbing, without backtracking, but this time you search from G to S.

Direction matters. Now hill climbing succeeds.

No branching on the goal node and the first node moving away from the goal node so you have:

$1 + 1 + 1 + 2 \times 4 = 11$ nodes in the search tree.

Many, for some reason, included only the nodes on the path from G to S, ignoring the losing nodes in the binary expansions, and lost one point.

Part E (3 points)

Still another friend suggests that you use bidirectional breadth-first search. That is, you expand alternately from the S and G, producing two search trees.

You encounter a common node after 3 levels of expansion on both sides.

The tree from S is binary: $\frac{2^{3+1}-1}{2-1} = 15$

The tree from S is unary for the first two levels, then binary: $1 + 1 + 2 = 4$

Thus, $15 + 4 - 1 = 18$ nodes total, counting the common node just once. Full credit even if you double counted the common node.

Part F (3 points)

Yet another friend suggests that you use branch-and-bound search so as to ensure an optimal path. What is the total number of nodes in the search tree?

Alas, branch-and-bound extends the shortest path until it finds that the shortest path leads to the goal. Thus, all 64 nodes end up in the search tree.

Part G (3 points)

Your boss orders you to use A* search from S to G so as to not only ensure an optimal path, but reduce the size of the search tree. What is the total number of nodes in the search tree?

Because of the curious layout of the nodes, you get no help from either dynamic programming or inclusion of the straight-line distance to the goal in the distance meansurements. All 64 nodes end up in the search tree again.

Part H (3 points)

Your boss then, after further thought, orders you to use A* search from G to S, and claims that the search tree will be much smaller than if you were to use A* from S to G.

This time the straight-line distance to the goal eliminates a great deal of work. For example, none of the left side nodes would end up in the search tree, because you cannot get to them without going through S, which means that any path to any left side node certainly would be longer than a path to S.

Problem 4

We use the following notation:

S_i	are the symptom variables, e.g., `cough, fever, painful toe`
D_i	are the disease variables, with values as given in the problem
`May-cause`(D_i, S_j)	disease i may cause symptom j

Part A

The domain of values of each symptom variable is the set of diseases that may cause it.

$$S_j = \{D_i \mid \text{may-cause}(D_i, S_j)\}$$

Intuitively, the value of a symptom variable is that disease that is said to be causing it, in this representation.

Part B

Constraints among disease variables must assure that the same "disease slot" is not assigned to more than one disease variable.

$$\text{Constraint}(D_i, D_j) = \begin{cases} \text{true} & \text{if } D_i = \texttt{NOTPRESENT} \text{ or } D_j = \texttt{NOTPRESENT} \text{ or } D_i \neq D_j \\ \text{false} & \text{otherwise} \end{cases}$$

Part C

There are no constraints among symptom variables.

Part D

To assure that each symptom has an explanation, we insist that if the value of a symptom variable is one of the diseases (which must be linked to it by a `may-cause` relationship), then the value of the corresponding disease variable may not be `NOTPRESENT`.

Note that this is a bit subtle, but has the right behavior. We cannot insist that *every* cause of a symptom be present, but this constraint assures that whichever one is chosen (by the value of the symptom variable) as the cause is in fact not `NOTPRESENT`—in other words, will have one of the DX_i values assigned to it.

The representation that we have chosen here allows us to express the constraints among symptoms and diseases as purely binary constraints, which is what PS5 used. A few other representations were almost correctly explored by a small number of students, but their defect is that they require multi-way constraints among a symptom variable and its possible causes. For example, one variant assigned just `PRESENT` as the value of all symptom variables, and then enforced a constraint that insisted that at least one of its `may-cause` diseases was not `NOTPRESENT`. This test cannot be accomplished by our code.

Part E

Recall that the diagnostic strategy begins with a single disease, DX_1. When `tuberculosis` is assigned this value, constraint propagation eliminates DX_1 as a possible value for all other diseases, thus leaving both `flu` and `infected toe` with values `NOTPRESENT`. The `NOTPRESENT` value at `flu` eliminates `flu` as a possible value for `cough`, which must therefore be `tuberculosis`. The `NOTPRESENT` values at `flu` and `infected toe` also eliminate those possible values for `fever`, which therefore has no possible values. This is a contradiction, and will cause the searcher to abandon the solution path that assigns DX_1 to `tuberculosis`.