Recitation 1, Thursday, September 11 and September 12

Rules Problem 1

Part A: Forward chaining

You need book recommendations for two of your friends, so you decide to use your forward-chaining book recommender.

Here’s what you know.

Initial Database of assertions:
1 (Max lives-in WashingtonDC)
2 (Jane lives-in SanFrancisco)
3 (Max likes science-fiction)
4 (Jane likes PhilipKDick)
5 (Pat likes TheThreeStigmataOfPalmerEldritch)
6 (PhilipKDick is-author-of Ubik)
7 (PhilipKDick is-author-of TheManInTheHighCastle)
8 (PhilipKDick is-author-of ThePenultimateTruth)

Note. The rule syntax below is simplified and different from the syntax if you use Python, as described in the problem set. In the problem set, Rule R1 would be written as:
R1 IF( AND( '(?x) like PhillipKDick' ),
    THEN( '(?x) likes science-fiction' ))

Rules:
R1 IF (?x likes PhilipKDick)
    THEN (?x likes science-fiction)

R2 IF (?x likes Ubik)
    THEN (?x likes alternate-realities)

R3 IF (AND (?x lives-in SanFrancisco)
    (?x likes science-fiction))
    THEN (?x likes alternate-realities)

R4 IF (?x lives-in WashingtonDC)
    THEN (?x likes politics)

R5 IF (AND (?x likes politics)
    (?x likes science-fiction))
    THEN (ThePenultimateTruth is-recommended-for ?x)

R6 IF (?x likes alternate-realities)
    THEN (TheManInTheHighCastle is-recommended-for ?x)
Fill out the following table to show the details of running the forward chainer. Assume that by "Matched Rules" we mean only those whose firing would add new assertions to the database. Use rule ordering for the conflict resolution strategy. Assume new assertions are added after already existing ones. Terminate when no further assertions can be made. You may abbreviate clauses as long as there is no ambiguity. (Note: There may be more lines in the table than you need.)

<table>
<thead>
<tr>
<th>Step</th>
<th>Matched Rule(s)</th>
<th>Rule Instance Binding(s)</th>
<th>Rule Fired</th>
<th>Database Assertion(s) Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part B: Backward chaining
One of your friends suggests that Pat might like TheManInTheHighCastle, but you want your backward chainer to help you prove whether or not that statement is true.

You use the same assertions as in your forward chaining system, plus a new assertion:
(Pat lives-in SanFrancisco)

You also use the same six rules as in your forward chaining system, plus a new rule:

R7 IF (?x likes TheThreeStigmataOfPalmerEldritch)
   THEN (?x likes PhilipKDick)

You then ask your backward chainer to prove the following assertion:
(TheManInTheHighCastle is-recommended-for Pat)

Using this assertion as the root node, draw the goal (AND/OR) tree that your system uses to prove the assertion. (The root node is provided below.) Assume that your system uses rule ordering as a conflict resolution strategy. Also assume that if an assertion cannot be proven via rules or existing assertions, it fails. (In other words, your system does not query you for an answer.) Label each branch of the tree with the name of the rule (e.g., R1) that it represents.

(TheManInTheHighCastle is-recommended-for Pat)