Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2003
Recitation 2, September 11/12

Rules Rule Prof. Bob Berwick

Agenda

1. **Intelligence = "Knowledge or Rule Based System" = Knowledge + Search (**What & How)
2. **Principles of Rule Based Systems :**
   a. ***How*** is *knowledge structured*? (REPRESENTATION); How *much knowledge* do we need?
   b. ***What*** can we do with this knowledge: How do we *recognize alternative knowledge*? (MATCHING)
   c. How do we *SEARCH* among alternatives? (forward and backward chaining); rule conflict resolution
3. **Forward chaining** practice example (note: if it can't infer something, it *stops)*
4. **Backward chaining** practice example (note: if can't infer something, it *asks* the user)

1. Intelligence= Knowledge-Based Systems as Reasoning systems

- Knowledge-based = knowledge + search. *How* to represent knowledge *How* to do search. Important: *YOU CAN'T DO OR UNDERSTAND KNOWLEDGE ENGINEERING BY JUST TALKING ABOUT IT.*
- How to do search: essentially, *logic* (one rule: (Generalized) *modus ponens*) , viz., from an implication and the premise of the implication, you can infer the conclusion. ($\alpha \Rightarrow \beta$ and α, conclude β, e.g., Person implies mortal & Socrates is a person, conclude Socrates is mortal, an *inference*).

  Antecedent (If part): **Person** x
  Consequent (new assertion: **Mortal** x

3. Principles: representation (what is encoded); search, matching (how the representation is used)

But, two basic, different ways of doing this:

4. Move from assertions to conclusions: **Forward chaining**

- **Start with sentences in the knowledge base** and generate new conclusions that in turn can allow new inferences to be made. Usually used when we add a new fact to the knowledge database and we want to generate its consequences. Example: bagger system (add list of stuff to pack, and see what assertions are possible; note that we use *variables* here in matching.)

Below is a description of how the **forward chaining** system works for this class.
Given a set of rules,

1. For each rule, match its variables against all assertions in the database to find bindings that make all the antecedents of rule match assertions in the database (thereby making the rule true). Check that the firing of the rule would actually modify the database. If so, create a triggered rule instance. The result of this step is the set of all triggered rule instances.

2. Choose a triggered rule instance (e.g. the first one), and fire it; i.e. carry out its consequent. Note: A **deduction** system only adds assertions; a production system adds and deletes assertions.

3. Since the database has been changed, discard rest of triggered rule instances and repeat matching to create triggered rule instances, choosing triggered rule (conflict resolution), and firing a rule instance (and discarding the rest) until there are no more rule instances that can be created or until an explicit stop command is executed.

Note that if you always choose the first triggered rule instance, it's terribly inefficient to compute the entire set of triggered rule instances. If you use a different conflict resolution technique, however, you might very well need to look at the entire list. In class discussions and on exams, assume that all the triggered rule instances are first found, and then conflict resolution is applied)

Deduction Rule System vs Production Rule System

A deduction rule system will always converge to the same result except in the case of STOP assertions and infinite loop situations. (STOP assertions generally are not considered part of deduction systems.) What this means is that in a deduction system, you could fire any number of triggered rules before rematching rules with assertions, as long as you have a mechanism for making sure that the same assertion isn't added to the database. In the forward chainer for this class, the checking for rules that would add existing assertions is done *before* the triggered rule instances are created, so only one rule is fired at a time, i.e. one triggered rule instance is executed, then the forward chaining process starts over.

A production rule system will not always converge to the same result if the conflict resolution technique introduces randomness into the order of rule execution. Production rules can add explicit STOP assertions, which stop execution of the chainer, or delete assertions that would cause other rules to trigger. Thus the order of rule execution matters in these systems.

5. **Move from conclusions to assertions: Backward chaining**

- **Start with something we want to prove**, find implication sentences that would allow us to conclude it, and then attempt to establish their premises in turn. This is normally used when we want a goal to be proved. ("Bottom-up" inference – must use *variables* in *both* the goal and the rules to try *tentative* (guessed) assignments.) Example: Slagle integration. (NOTE you can do this via forward chaining as well, but there are differences for how the system behaves….) Pseudo-code for backchaining:

```
Rule form:  IF <premise> THEN <conclusion> where <premise> is a list C of
clauses.
1. Find the list of rules that conclude about the
current attribute A (unification may be required)
2. If there are no such rules,
ask the user for the value of the attribute A.
3. Otherwise, for each rule R, use the following to determine the truth of the
premise of the rule:
For each clause C in the premise of R:
      3.1 Look in the DB for the value of the attribute mentioned in C.
      3.2 If there is no information in the database about that attribute,
           [recursion step: take another step back in the inference chain –
           creates AND/OR Tree]:
      3.3 Start from the top with that attribute as the current one.
Evaluate the truth of C using the information in the database
4. If the premise of the rule evalutes to "true" make the conclusion shown given in
the rule (set the attribute value).
```

For this class we will always assume that our backward chainers are trying to prove the truth of a conclusion (also called a goal or hypothesis) by first checking the database for matching assertions, then checking for rules that have the hypothesis as a consequent, then asking the user. Note that we assume that the system only asks the user questions if there are no rules which have the desired conclusion as a consequent. In other words, if there are rules that would produce the conclusion, then the system assumes that if they don't produce the conclusion, then the conclusion must be false.

PRACTICE PROBLEMS 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.

Database of assertions:
(Max lives-in WashingtonDC)
(Jane lives-in SanFrancisco)
(Max likes science-fiction)
(Jane likes PhilipKDick)
(Pat likes TheThreeStigmataOfPalmerEldritch)
(PhilipKDick is-author-of Ubik)
(PhilipKDick is-author-of TheManInTheHighCastle)
(PhilipKDick is-author-of ThePenultimateTruth)

Rules:

R1 if (?x likes PhilipKDick)
then (?x likes science-fiction)

R2 if (?x likes Ubik)
then (?x likes alternate-realities)

R3 if (?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 (?x likes politics)
then (ThePenultimateTruth is-recommended-for ?x)

R6 if (?x likes alternate-realities)
then (TheManInTheHighCastle is-recommend-for ?x)

Fill out the following table to show the details of running the forward chainer. 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.)

Step	Triggered Rule(s)	Rule Instance Binding(s)	Rule Fired	Database Assertion(s) Added
1				
2				
3				
4				
5				
6				
7				

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 6 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-recommend-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, that 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-recommend-for Pat)