MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence
Fall, 2010

Recitation 1, Thursday September 16

Forward and Backward Chaining Notes

Prof. Bob Berwick, 32D-728

Forward Chaining

You can think of the forward chaining process as that of filtering a set of rules to find the one that is applicable, then firing the rule, i.e., carrying out that rule's consequent, to change the state of the world. The state of the world is represented as a set of assertions, which are statements about what is true in the world. You might want to think about how 'deeply' the rules actually encode the state of knowledge about a particular situation, for example, the grocery bagging rules. How robust or fragile are they? How could you improve them?

Below is a brief synopsis of the forward chaining system we'll discuss today.

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 a 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, the rule is not redundant, so create a matched rule instance. The result of this step is the set of all matched rule instances. If this set is empty, stop.

2. Choose a matched rule instance (e.g., the first one), and fire it. Note that if there is more than one such matched rule that changes the database, one must have some conflict resolution method for picking the rule to fire; for example, one might pick the first rule in the system's list of rules, or the last rule, or the most recently used rule, or… (think of some other methods…how else could there be 'multiple matches' to resolve?). Note: A deduction system only adds assertions; a production system can both add and delete assertions. (So in class Winston called production systems 'general programming languages' – meaning that they can do anything that a general-purpose computer can do.)

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

Note that if you always choose the first matched rule instance, it's terribly inefficient to compute the entire set of matched rule instances. If you use a different conflict resolution technique, however, you might very well need to look at the entire list. Here we assume that all the matched rule instances are computed unless otherwise specified.

Deduction Rule Systems vs. Production Rule Systems

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 matched rules before re-matching 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 chaining system for this class, the checking for redundant rules, which would add already-existing assertions, is done before the matched rule instances are created, so only one rule is fired at a time, i.e., one matched rule instance is executed. Then the forward chaining process starts all over again.

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 rule systems can add explicit STOP assertions, which stop execution of the chainer, or DELETE assertions that would cause other rules to be matched.

A rule with a consequent that stops the chaining is an example of representing knowledge about the problem-solving process itself, a kind of 'meta-knowledge.' You might contrast this sort of rule with ones that represents knowledge about the state of the world.

Backward Chaining

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. We make the following assumptions about the way this works:

1. The backward chainer first tries to find a matching assertion in the list of assertions (Principle AWP – ask me what this abbreviation stands for.)
2. If no matching assertion is found, the backward chainer tries to find a rule with a matching consequent.
3. If no matching consequent is found, the backward chainer assumes the hypothesis is false (we might alternatively have the system them ask the user whether the hypothesis is true, but we do not do that here).

Note that:

- The backward chainer tries rules in the order they appear in the database of rules.
- Antecedents are tried in the order they appear in the rules.
- When backward chaining, a NOT clause matches if and only if there is no matching assertion in the list of assertions, and rules that connect assertions in the list of assertions to the NOT clause.

Question: what about rule conflict resolution in this situation?

Forward vs Backward Chaining

Many rule systems can chain either forward or backward, but which direction is better? There are several rules-of-thumb.

Most importantly, you want to think about how the rules relate facts to conclusions. Whenever the rules are such that a typical set of facts can lead to many conclusions, your rule system exhibits a high degree of "fan-out," and argues for backward chaining. Whenever the rules are such that a typical hypothesis (i.e., a goal to be proven) can lead to many questions, your rule system exhibits a high degree of "fan-in" and argues for forward chaining. However, there are other considerations:

1. If the facts that you have or may establish may lead to a large number of conclusions, but the number of ways to reach the particular conclusion in which you are interested is small, then there is more fan-out than fan-in, and you should typically use backward chaining.

2. If the number of ways to reach the particular conclusion in which you are interested in is large, but the number of conclusions that you are likely to reach is small, then there is more fan-in than fan-out, and you should typically use forward chaining.

These are just rules-of-thumb. In many situations, however, neither fan-out nor fan-in dominates, leading to other considerations:

3. If you have not yet gathered any facts, and you are interested in only whether one of many possible conclusions is true, use backward chaining.

4. If you already have in hand all the facts you are ever going to get, and you want to know everything that can be concluded from those facts, use forward chaining.