G53KRR: Forward chaining, production systems

Forward chaining (for propositional Horn clauses):

input: a finite list of atomic sentences, $q_1, \ldots, q_n$

output: YES if KB entails all of q_i, NO otherwise

1. if all goals q_i are marked as solved, return YES
2. check if there is a clause $[p, \neg p_1, \ldots, \neg p_m]$ in KB, such that all of $p_1, \ldots, p_m$ are marked as solved and p is not marked as solved
3. if there is such a clause, then mark p as solved and go to step 1.
4. otherwise, return NO.

Another way to look at it:

input: a finite list of atomic sentences, $q_1, \ldots, q_n$

output: YES if KB entails all of q_i, NO otherwise

1. if all goals q_i are in KB, return YES
2. check if there is a clause $p_1 \wedge \ldots \wedge p_m \supset p$ in KB, such that all of $p_1, \ldots, p_m$ are in KB and p is not in KB
3. if there is such a clause, then add p to KB and go to step 1.
4. otherwise, return NO.

For first-order case, similar, but need to unify the pattern in the body of the rule $(P_1(\bar{x}) \wedge \ldots \wedge P_m(\bar{x}))$ with unit clauses $P_i(\bar{a})$ in KB first, and then apply the same substitution to $P(\bar{x})$.

For first-order case we do the inference from

$\forall x_1 \forall x_2 (Parent(x_1, x_2) \wedge Male(x_1) \supset Father(x_1, x_2))$, $Parent(bob, chris)$, $Male(bob)$

to $Father(bob, chris)$.

Production rule systems are forward-chaining reasoning systems which use production rules. Usually production rules are more complex than just Horn clauses (similar to how Prolog has more than just Horn clauses, also negation as failure etc.) but we will look at just Horn clauses. So this is different from the version in the textbook (much simpler).

We just assume that the knowledge base consists of: Working Memory WM which is a finite set of ground atoms (facts, or Working Memory Elements), and a finite set of production rules (universal Horn clauses).

The simplest way of reasoning would be to chain forward as above, but this may flood WM with a lot of irrelevant facts. Instead most production rule systems compute a *conflict set*: the set of all possible *rule instances* applicable for the current state of working memory. A rule instance is a substitution which

makes a pattern in some rule match the working memory elements, together with the rule itself. A *conflict resolution strategy* is used to determine which of the rule instances in the conflict set will actually be fired (which conclusions added). Most conflict resolution strategies pick a single rule instance based on one or more of the following criteria: specificity of the rules (which rule has a more specific pattern in the body); or the order in which rules appear in the program; or the order in which facts were added to working memory (for example, depth first where rule instances involving more recent facts are preferred) etc.

Exercise For the following production system, trace the results, assuming that the conflict resolution strategy is: an instance of most important applicable rule is selected. If there are more than one such instances, the instance is selected randomly. The order of rule importance is: R3 more important than R1, R1 is more important than R2.

F1 $animal(tiger)$

F2 $animal(cat)$

F3 $large(tiger)$

F4 $eatsMeat(tiger)$

F5 $eatsMeat(cat)$

R1 $\forall x(animal(x) \wedge large(x) \wedge eatsMeat(x) \supset dangerous(x))$

R2 $\forall x(animal(x) \supset breathesOxygen(x))$

R3 $\forall x(dangerous(x) \supset runAwayNow)$