32002: AI (Complexities of Resolution, Rule based system) **Spring 2014**

Lecture 12: January 29, 2014

Lecturer: K.R. Chowdhary *: Professor of CS (GF)*

Complexity of Resolution Proof

The question is, how you can be so clever to pickup the right clauses to resolve? The answer is that you take advantage of two ideas:

1. You can be sure that every resolution involves the negated theorem, directly or indirectly.
2. You know where you are and where you are going, hence you can note the difference to help you proceed with your intuition for selection of clauses.

Consider there are total n clauses, $c_1 \dots c_n$. We can try to match c_1 with $c_2 \dots c_n$, and in next level c_2 is matched with $c_3 \dots c_n$, and so on. This results to breadth first search (BFS). Consider that resolvents generated due to this matching are $c_1^{'} \dots c_m^{'}$. Next all the newly generated clauses are matched with the original, and then they are merged into the original. This process is repeated until contradiction is reached, which shows that theorem is proved. Since, the entire set of clauses are compared, the proof is bound to result, if it exists, at all. This gives completeness to the resolution proof.

The other alternative is, nodes which are farther and farther away are matched before those which are closer to the root. The c_1 is matched with first child c_2 out of $c_2 \dots c_n$. Then c_2 is matched with its first child generated, and so on, resulting to the search process called DFS (depth first search).

All resolution search strategies are subject to a version of *halting problem*, for search is not guaranteed to terminate unless there actually is a proof. In fact, all complete proof procedures for the first order predicate calculus are subject to halting problem.

12.1 Rule Based Reasoning

A rule is in the form of *if-then* $(p \rightarrow q)$, where p is called precondition or premises, or assertion and q is called goal or hypothesis, or consequence. For inference to be carried out the rules are chained. There are two ways they can be chained: the *forward-chaining* starts from the assertions and *back-ward* chaining from hypotheses. Rule-based systems (RBSs) constitute the one of the most common and simple to implement means for codifying the problem-solving know-how of human experts. Experts tend to express most of their problem-solving techniques in terms of a set of *situation-action* rules, and this suggests that RBSs should be the method of choice for building knowledge-intensive expert systems. The RBSs share following key properties:

1. They incorporate practical human knowledge in conditional *if-then* rules,
2. their skill increases at a rate proportional to the enlargement of their knowledge bases,

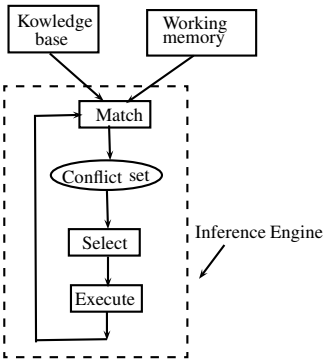

Figure 12.1: Inference Cycle.

3. they can solve a wide range of possibly complex problems by selecting relevant rules and then combining the results in appropriate ways,
4. they adaptively determine the best sequence of rules to execute, and
5. they explain their conclusions by retracing their actual lines of reasoning and translating the logic of each rule employed into natural language.

12.2 Inferencing using RBSs

An RBS consists of a knowledge base and an inference engine (figure 12.1). The knowledge base contains rules and facts. Rules always express a conditional, with an antecedent and a consequent component. The interpretation of a rule is that if the antecedent can be satisfied the consequent can too. When the consequent defines an action, the effect of satisfying the antecedent is to schedule the action for execution. When the consequent defines a conclusion, the effect is to infer the conclusion.

Several key techniques for organizing RBSs have emerged. Rules can be used to express deductive knowledge, such as logical relationships, and thereby to support inference, verification, or evaluation tasks. Conversely, rules can be used to express goal-oriented knowledge that an RBS can apply in seeking problem solutions and cite in justifying its own goal-seeking behavior. Finally, rules can be used to express causal relationships, which an RBS can use to answer "what if" questions, or to determine possible causes for specified events.

In a rule based system, each *if* pattern may match to one or more of assertions in a collections of assertions. The collections of assertions is called *working-memory* (figure 12.1). The *then* patterns are assertions to be placed into the working memory. A system like this is called *deduction system*. An if-then rule may be like,

```
Ri:   if   if1
           if2
            ...
     then  then1
           then2
          ...
```

where R_i is name of the rule, or rule number. The assertions, i.e., working memory, matches with one or more if patterns, like if1, if2, etc. A more realistic example of a rule is:

```
R1: if 1. stiff neck and
       2. high temperature
       3. impairment of conciousness occur togetehr,
    then
          meningitis is suspected.
```

In the above, meningitis is "Inflation of membrane of spinal chord and brain."

Facts - the other kind of data in a knowledge base, express assertions about properties, relations, propositions, etc. In contrast to rules, which the RBS interprets as imperatives, facts are usually static and inactive implicitly, a fact is silent regarding the pragmatic value and dynamic utilization of its knowledge. Although in many contexts facts and rules are logically interchangeable, in the RBSs they are quite distinct.

In addition to its static memory for facts and rules, an RBS uses a *working-memory* to store temporary assertions. These assertions record earlier rule-based inferences. We can describe the contents of working memory as problem-solving state information. Ordinarily, the data in working memory adhere to the syntactic conventions of facts. Temporary assertions thus correspond to dynamic facts.

The basic function of an RBS is to produce results. The primary output may be - a problem solution, an answer to a question, or an analysis of some data. Whatever the case, an RBS employs several key processing determining its overall activity. A *world* manager maintains information in working memory, and a built-in control procedure defines the basic high-level loop; if the built-in control provides for programmable specialized control, an additional process manages branches to and returns from special control blocks.

Some times, the patterns specify the *actions* rather than *assertions*, e.g., "to put them on the table". In such case the the rule based system is called *reaction system*.

In both the deduction systems and reaction systems, forward chaining is the process of moving from the *if* patterns to *then* patterns, where *if* patterns identifies the appropriate situation for deductions of new assertion, and performance of an action in the case of reaction system.

References

[1] Chowdhary K.R. (2020) Logic and Reasoning Patterns. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. https://doi.org/10.1007/978-81-322-3972-7_4