CS323: AI (Procedural Interpretation of Clausal form expressions) **Spring 2015**

Lecture 6: January 16, 2015

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

6.1 Procedural Interpretation

It is easy to procedurally interpret the sets of clauses, say, **A**, which contain at most one positive literal per clause. However, along with this any number of negative literals can also exist. Such sets of clauses are called *Horn sentences* or *Horn Clauses* or simply clauses. We distinguish three kinds of *Horn clauses.*

1. '[]'the *empty clause*, containing no literals and denoting the truth value *false*, is interpreted as a *halt statement.*
2. $\bar{B}_1 \vee \cdots \vee \bar{B}_n$, a clause consisting of no positive literals and $n \geq 1$ negative literals, is interpreted as a *goal statement.* Note that goal statement is negated and added into the knowledge base to obtain the proof through *resolution refutation.*
3. $A \vee \bar{B}_1 \vee \cdots \vee \bar{B}_n$, a clause consisting of exactly one positive literal and $n \geq 0$ negative literals is interpreted as a *procedure declaration* (i.e., rule in Prolog program). The positive literal A is the *procedure name* and the collective negative literals are the *procedure body.* Each negative literal B_i, in the procedure body is interpreted as a *procedure call.* When $n = 0$ the procedure declaration has an empty body and interpreted as an unqualified assertion of fact.

In the procedural interpretation, a set of procedure declarations is a program. Computation is initiated by an *initial goal* statement, which proceeds by using declared procedures to derive new goal statements (*subgoals*) B_is from old goal statements, and terminates on the derivation of the halt statement. Such derivation of goal statements is accomplished by *resolution*, which is interpreted as *procedural invocation.*

Consider that, a selected procedure call $\bar{A}_1$ inside the body of a goal statement as,

$$\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{A}_i \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n \tag{6.1}$$

and a procedure declaration is given as,

$$A' \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m, m \geq 0. \tag{6.2}$$

Suppose, the name of procedure A' matches with the procedure call A_i, i,e., some substitution θ of terms for variables makes A_i and A' identical. In such a case, the resolution derives a new goal statement by disjunction formulas (6.1) and (6.2) as given below, subject to substitution θ.

$$(\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n)\theta. \tag{6.3}$$

In general, any *derivation* can be regarded as a computation, and any *refutation* (i.e. derivation of []) can be regarded as a successfully terminating computation. It is to be noted that, only goal oriented resolution derivations correspond to the standard notion of computation.

Thus, a goal-oriented derivation, from an initial set of Horn clauses $\mathbf{A}$ and from an initial goal statement (computation) $C_1 \in \mathbf{A}$, is a sequence of goal statements $C_1, \ldots, C_n$. So that each C_i contains a single selected procedure call and C_{i+1}, obtained from C_i by procedure invocation relative to the selected procedure call in C_i, using a procedure declaration in $\mathbf{A}$.

For the implementation of above, one method is *model elimination.* Using this, the selection of procedure calls is governed by the last-in/first-out rule: a goal statement is treated as a stack of procedure calls. The selected procedure call must be at the top of the stack. The new procedure calls which by procedure invocation replace the selected procedure call are inserted at the top of the stack. This would result to a *depth-first search* procedure.

The Predicate logic is a *nondeterministic* programming language. Consequently, given a single goal statement, several procedure declarations can have a name which matches the selected procedure call. Each declaration gives rise to a new subgoal statement. A *proof procedure* which sequences the generation of derivations in the search for a refutation behaves as an *interpreter* for the program incorporated in the initial set of clauses.