32002: AI (First Order Predicate Logic, Interpretation and Inferences) **Spring 2015**

Lecture 4: January 12, 2015

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

4.1 First Order Predicate Logic

First-order logic is best suited as a basic theoretical instrument of a computer theorem proving program. From the theoretical point of view, however, an inference principle need only be *sound* (i.e., allow only logical consequences of premises to be deduced from them) and *effective* (i.e., it must be algorithmically decidable whether an alleged application of the inference principle is indeed an application of it).

The *resolution principle* is machine-oriented rather than human-oriented. The resolution principle is quite powerful, both in the psychological sense that it condones single inferences which are often beyond the ability of the human to grasp, and in the theoretical sense that it alone, as sole inference principle, forms a complete system of first-order logic.

4.2 Syntax and Semantics

The Syntax deals with the formal part of language in abstraction from its meaning. It concerns with the definition of *well-formed formulas*. Syntax in its narrow sense and also deals with the study of axioms, rules of reference and proofs, which constitute proof theory. Semantics is concerned with the interpretation of language and includes such notions as meaning, logical implication and truth.

It is convenient to restrict attention to predicate logic programs written in *clausal form*. Such programs have an especially simple syntax but retain all the expressive power of the full predicate logic.

Atomic formula. A string of symbols consisting of a predicate symbol of degree $n \geq 0$ followed by n terms is an *atomic formula*.

Clause. A clause is a disjunction $L_1 \vee \cdots \vee L_n$ of literals L_i, each of which is *atomic formula* $P(t_1, \ldots, t_m)$ or the negation of atomic formulas, where P is a predicate symbol and t_i, are *terms*. A finite set (possibly empty) of literals is called a clause. The empty clause is denoted by: []

Sentence. A sentence is a finite set of clauses.

Literals. An atomic formula is a literal; and if A is an atomic formula then $\neg A$ is a literal.

Term. A term is either a variable or an expression like $f(t_i, \ldots, t_n)$, where f is a function symbol and t_i are terms. Constants are $0 - ary$ function symbols. A variable is a term, and a string of symbols consisting of

a function symbol of degree $n \geq 0$ followed by n terms is a term.

A set of clauses $\{C_1 \ldots, C_n\}$ is interpreted as the conjunction, C_1 and $\ldots$, and C_n. A clause C containing just the variables $x_1, \ldots, x_n$ is regarded as universally quantified, for example,

$$for\ all\ x_1, \ldots, x_n C \tag{4.1}$$

Ground literals. A literal which contains no variables is called a ground literal.

Ground clauses. A clause, each member of which is a ground literal, is called a ground clause. In particular [] is a ground clause.

4.3 Representation in Predicate Logic

The statements of FOPL are flexible enough to permit the accurate representation of *natural languages.*

- English sentence: Ram is man and Sita is women.

  Predicate form: $man(Ram) \wedge woman(Sita)$

- English sentence: Ram is married to Sita.

  Predicate form: $married(Ram, Sita)$

- English sentence: Every person has a mother.

  Reorganized form of above: For all x, there exists a y, such that if x is person then x's mother is y.

  Predicate form: $\forall x \exists y[person(x) \Rightarrow hasmother(x, y)]$

- English sentence: If x and y are parents of a child z, and x is man, then y is not man.

  $\forall x \forall y[[parents(x, z) \wedge parents(y, z) \wedge man(x)] \Rightarrow \neg man(y)]$

We note that predicate language comprises constants {Ram, Sita}, variables $\{x, y\}$, operators $\{\Rightarrow, \wedge, \vee, \neg\}$, quantifiers $\{\exists, \forall\}$ and functions/ predicates {married(x, y), person(x)}. Unless specifically mentioned, the letters $a, b, c, \ldots$ at the beginning of English alphabets shall be treated as constants to indicate names of *objects* and *entities*, and those at the end, i.e., $u, v, w, \ldots$ shall be used as variables or identifiers for objects and entities.

To indicate that an expression is universally true, we use the *universal quantifier* symbol $\forall$, meaning 'for all'. Consider the sentence "any object that has a feathers is a bird." Its predicate formula is: $\forall x[hasfeathers(x) \Rightarrow isbird(x)]$. Then certainly, $hasfeathers(parrot) \Rightarrow isbird(parrot)$ is true. Some expressions, although not always *True*, are *True* at least for some objects: in logic, this is indicted by 'there exists', and the *existential quantifier* symbol $\exists$ is used for this. For example, $\exists x[bird(x)]$, when *True*, this expression means that there is at least one possible object, that when substituted in the position of x, makes the expression inside the parenthesis as *True.*

Following are some examples of representations of knowledge FOPL.

Example 4.1 *Kinship Relations.*

Solution.

$mother(namrata, priti)$.(That is, Namrata is mother of Preeti.)
$mother(namrata, bharat)$.
$father(rajan, priti)$.
$father(rajan, bharat)$.
$\forall x \forall y \forall z[father(y, x) \wedge mother(z, x) \Rightarrow spouse(y, z)]$.
$\forall x \forall y \forall z[mother(z, x) \wedge mother(z, y) \Rightarrow sibling(x, y)]$.

□

Example 4.2 *Represent the following sentences by predicate calculus wffs.*

1. A computer system is intelligent if it can perform a task which, if performed by a human, requires intelligence.

Ans.

$$\exists x[[(perform(human, x) \rightarrow requires(human, intelligence)) \wedge (perform(computer, x)] \rightarrow intelligent(computer))]$$

2. If a program cannot be told a fact, then it cannot learn that fact.

Ans. $\forall x[(program(x) \wedge \neg told(x, fact)) \rightarrow \neg learn(x, fact)]$

Example 4.3 *Blocks world.*

Consider that there are are physical objects, like - *cuboid*, *cone*, *cylinder* placed on the table-top, with some relative positions, as shown in figure 4.1. There are four blocks on the table:a, c, d are cuboid, and b is a cone. Along with these there is a robot arm, to lift one of the object having clear top.

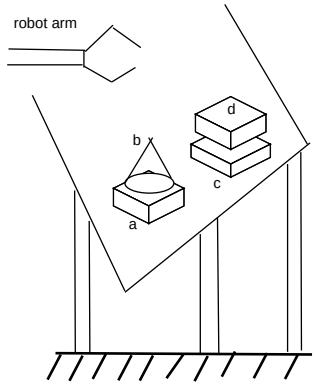

Figure 4.1: Blocks World.

Solution. Following is the set of statements about the blocks world (called knowledge base):

$$
\begin{aligned}
&cuboid(a).\\
&cone(b).\\
&cuboid(c).\\
&cuboid(d).\\
&onground(a).\\
&onground(c).\\
&ontop(b,a).\\
&ontop(d,c).\\
&toplcear(b).\\
&topclear(d).\\
&\exists x \exists y[topclear(x) \wedge toplcear(y) \wedge \neg cone(x) \Rightarrow puton(y,x)]
\end{aligned}
$$

The knowledge specified in the blocks world indicate that objects a, c, d are cuboid, b is cone, a, c are put on the ground, and b, d are on top of a, c, respectively, and the top of b, d are clear. These are called facts in knowledge representation. At the end, the rule says that there exists objects x and y such that both have their tops clear and x is not a cone, then y can be put on the object x. □

4.3.1 Bound and Free Variables

A variable in a wff is bound if it is within the scope of a quantifier naming the variable, otherwise the variable is free. For example, in $\forall x(p(x) \rightarrow q(x,y))$, x is bound and y is free; in $\forall x(p(x) \rightarrow q(x)) \rightarrow r(x)$, the x in $r(x)$ is free variable. In the latter case it is better to rename the variable to remove the ambiguity, hence we rephrase this statement as $\forall x(p(x) \rightarrow q(x)) \rightarrow r(z)$. An expression can be evaluated only when all the variables in that are bound.

4.4 Interpretation and Inferences

A FOPL statement is made of predicates, arguments (constants or variables), functions, operators, and quantifiers. Interpretation is process of assignment of truth values (True/False) to subexpressions and atomic expressions, and computing the resultant value of any expression/ statement. A statement or expression in predicate logic is also called *wwf* (well formed formula).

Consider the interpretation of predicate formula:

$$\forall x[bird(x) \rightarrow flies(x)] \tag{4.2}$$

To find out the *satisfiability* of the formula (4.2), we need to substitute (*instantiate*) a value for x (an instance of x) and check if $flies(x)$ is true. Until, that x is found, it may require instantiation with large number of values. Similarly, to check if the equation (4.2) is valid, it may require infinitely large number of values in the domain of x to be verified. If any one of that makes the formula false, the formula is not valid.

Example 4.4 *Given, "All men are mortal" and "Socrates is man", infer using predicate logic, that "Socrates is mortal".*

Solution. The above statement can be written in predicate logic as:

$$\begin{aligned} &\forall x[man(x) \Rightarrow mortal(x)], \\ &man(socrates). \end{aligned} \tag{4.3}$$

Using a rule called *universal instantiation*, a variable can be instantiated by a constant and universal quantifier can be dropped. Hence, from 4.3 we have

$$\begin{aligned} &man(socrates) \Rightarrow mortal(socrates), \\ &man(socrates). \end{aligned} \tag{4.4}$$

Using the rule of *modus ponens* on 4.4 we deduce $mortal(socrates)$. It is also *logical consequence*. If $\Gamma = \{[man(socrates) \Rightarrow mortal(socrates)] \wedge man(socrates)\}$, and $\alpha = mortal(socrates)$, then we can say that $\Gamma \vdash \alpha$.

The set of formulas Γ is called knowledge-base. To find out the result for the query "Who is man?", we must give the query

$$?man(X).$$

in Prolog (to be discussed later), which will match (called *unify* or *substitute*) $man(X)$ with $man(socrates)$ with a unification set, say, $\theta = \{socrates/X\}$. The substitution which returns $man(socrates)$ is represented by $man(X)\theta$. □

Example 4.5 *Prolog Program.*

The sentence in equation (4.3) will appear in prolog as,

$$\begin{aligned} &mortal(socrates) :- man(socrates). \\ &man(socrates). \end{aligned}$$

where, sign ':-'is read as 'if'. The subexpression before ':-'is called 'head'or *procedure name* and the part after ':- 'is called *body* of the *rule*. The sentence (4.4) can also be written in a *clause form*, (to be precise, in *Horn clause* form) it becomes,

$$\begin{aligned} &mortal(socrates) \vee \neg man(socrates). \\ &man(socrates). \end{aligned} \tag{4.5}$$

□

4.4.1 The Procedural Interpretation

It is easy to procedurally interpret the sets of clauses 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*.

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*) 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,

$$\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{A}_i \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n \tag{4.6}$$

and a procedure declaration is given as,

$$A' \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m, m \geq 0. \tag{4.7}$$

Suppose, the name of procedure matches with the procedure call, 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 (4.6) and (4.7) as given below, subject to substitution θ.

$$(\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n)\theta. \tag{4.8}$$

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.