INST0072 Logic and Knowledge Representation

Lecture 2

Prolog: Facts, Queries and Rules

In the Last Lecture

In the last lecture we saw that:

- The semantics (meaning) of a propositional formula is defined in terms of its *truth table*. Each row in the truth table corresponds to an *interpretation*. An interpretation gives a value of 'true' or 'false' to each of the propositions in the logic's *signature* (vocabulary of propositions).

- An interpretation *satisfies* a formula if it assigns a value of 'true' to the formula, in which case the interpretation is a *model* of the formula. A formula is *satisfiable* if it has at least one model. Otherwise it is *unsatisfiable*.

- The formula A *entails* the formula B if every model of A is also a model of B.

- An *inference rule* is a rule for generating a new formula (called the *conclusion*) from a list of existing formulas (called the *premises*). A set of inference rules is called a *calculus*, and a series of applications of the inference rules is called a *derivation*.

Prolog Facts

- *Facts* in Prolog state that certain propositions are true or that certain relationships hold between certain objects.

- For example, the fact

```
it_is_raining.
```

  states that the proposition 'it_is_raining' is true, and the fact

```
likes(john, big_mac).
```

  states that the relation 'likes' holds between the objects 'john' and 'big_mac'.

- In Prolog and in logic programming in general, we generally call propositions and relationships such as those above *predicates*, while the objects such as 'john' and 'big_mac' above are called *arguments*. The *arity* of a predicate is the number of arguments it takes, so 'it_is_raining' has arity 0 whereas 'likes' has arity 2.

An Example Program of Facts

- Suppose we have the following Prolog program (which is really just a list or database of facts):

```
likes(john, big_mac).      /* comments in Prolog programs ar
likes(mary, big_mac).      /* written between backslash and
likes(mary, ice_cream).    /* symbols like this.
likes(sue, apple_pie).
likes(sue, ice_cream).     % Single-line comments can also s
likes(chris, milkshake).   % with a percentage sign.
```

  We regard this set of facts as the *definition* for the predicate 'likes' - more properly the definition of the predicate 'likes/2', the '/2' indicating the number of arguments 'likes' requires (i.e. the arity of the predicate).

- If the program above is saved in a file called 'foodPreferences.pl' (a plain text file), we can load it into SWI Prolog with this *consult* command (assuming it is in the default or current SWI Prolog directory):

```
?- [foodPreferences].
```

  or equivalently by using the SWI Prolog '*file - consult ...*' menu option.

[foodPreferences.pl]

Prolog Queries

- Having defined a predicate (e.g. 'likes/2'), we can use this definition to find out whether the predicate is true or false for certain values of its arguments by making a *query*.

- Thus if we ask (in SWI Prolog)

```
?- likes(sue, ice_cream).
```

  Prolog will answer 'true' (or 'yes' in some systems), since the fact

```
likes(sue, ice_cream).
```

  is in the program. ('?-' is a notation commonly adopted in Prolog systems to indicate that a query is being asked.)

- If we ask

```
?- likes(john, ice_cream).
```

  then Prolog will answer 'false' (or 'no' in some systems).

- Queries are also known as goals. Thus the first query above represents the goal 'likes(sue, ice_cream)'.

Queries With Variables

- We can also introduce variables into both facts and queries. Variables are used to stand for an unspecified single object. In SWI Prolog, variable names begin with an upper-case letter or the underscore character _.

- The following query means 'is there an X such that likes(chris, X) is true?':

```
?- likes(chris, X).
```

- Since 'likes(chris, milkshake)' appears in the program, the answer is 'true', together with the *substitution* {X=milkshake}.

- When more than one substitution makes a query true, we can view these one-by-one in SWI Prolog by typing ';'. For example:

```
?- likes(sue, X).
```

- Note that we can also ask queries such as:

```
?- likes(X, big_mac).
?- likes(X,Y).
```

Conjunctions of Queries

- We can also extend our queries from those expressing a single goal to those expressing a conjunction of goals (i.e. an "and" of two or more goals).
- An "and" in Prolog is indicated with a comma. For example, the query

```
?- likes(john, X), likes(mary, X).
```

  is asking for an X such that both john and mary like X.
- In this case SWI Prolog first outputs

```
X = big_mac
```

  and then, if we type ';' for an alternative substitution, outputs

```
false
```

  because the program does not list anything else that both john and mary like.

A Family Program

The program 'mothersAndFathers.pl' contains definitions for 'father/2', 'mother/2', 'male/1' and 'female/1'.

```
father(geoffrey, sylvia).
father(geoffrey, katherine).
father(geoffrey, rob).
father(geoffrey, andrew).
father(rob, rebecca).
    .....
mother(linda, oscar).
mother(linda, spencer).
mother(linda, lucy).
mother(katherine, bene).
    .....
male(oscar).
male(spencer).
    .....
female(rebecca).
female(natascia).
    .....
```

[mothersAndFathers.pl]

Prolog Rules

- Prolog *rules* provide a means of defining relationships in terms of other relationships. One of the rules in 'mothersAndFathers.pl' is

```
grandfather(X,Z) :- father(X,Y), father(Y,Z).
```

- The ':-' is read as an 'if', so that the above rule can be interpreted as

'X is the grandfather of Z if X is the father of Y and Y is the father of Z'

- In general, rules have the form

A :- B_1, ..., B_n.

'A' is known as the *head* of the rule, the conjunction 'B_1, ..., B_n' is known as the *body*.

- As another example, the formula on Slide 4 of Lecture 1 can be represented in Prolog as the rule

```
i_am_wet :- it_is_raining, i_am_outside.
```

- The definition of a predicate may be a mixture of facts and rules:

```
male(X) :-
    father(X, Y).
male(oscar).
.....
```