LOGIC PROGRAMMING

Lecture#3

Clauses and predicates

1. Clauses

- **Definition of Clause**:
  - Apart from comments and blank lines, which are ignored, a Prolog program consists of a succession of *clauses*.
  - *A clause can run over more than one line or* there may be several on the same line. A clause is terminated by a dot character.
  - There are two types of clause: *facts and rules*.

1. Clauses – Facts (cont.)

1. **Form of Facts**:
   - ***head***
   - ***head*** *is called the* ***head of the clause***.
   - *It takes the same form as a goal entered by* the user at the prompt, i.e. it must be an atom or a compound term.
   - Atoms and compound terms are known collectively as ***call terms***.

1. Clauses - Facts (cont.)

Example:

```
hello.

likes(sara, flowers).

tasty(chocolate).
```

1. Clauses - Rules (cont.)

1.2 Form of Rules:

head:-t_1,t_2, … , t_k. (k>=1)

- ***head*** *is called the head of the clause (or the head of the rule) and,*
- *as for facts,* must be a call term, i.e. an atom or a compound term.
- **:-** is called the neck of the clause (or the ***'neck operator'***). *It is read as 'if'.*
- t_1, t_2, …, t_k *is called the body of the clause (or the body of the rule). It specifies* the conditions that must be met in order for the conclusion, represented by the head, to be satisfied.

1. Clauses – Rules (cont.)

- The body consists of one or more components, separated by commas. The components are *goals and the commas are read as 'and'.*
- Each goal must be a call term, i.e. an atom or a compound term.
- A rule can be read as '*head is true if* t_1, t_2, …, t_k *are all true*'.
- The head of a rule can also be viewed as a *goal with the components of its body* viewed as subgoals.
- Another reading of a rule is 'to achieve goal *head, it is* necessary to achieve subgoals t_1, t_2, …, t_k *in turn*'.

1. Clauses – Rules (cont.)

- **Some examples of Rules are:**

```
large_animal(X):-animal(X),large(X).
grandparent(X,Y):-father(X,Z),parent(Z,Y).
go:-write('hello world'),nl.
```

1.Clauses (cont.)


```
/* Animals Program 2 */
dog(fido). large(fido).
cat(mary). large(mary).
dog(rover). dog(jane).
dog(tom). large(tom). cat(harry).
dog(fred). dog(henry).
cat(bill). cat(steve).
small(henry). large(fred).
large(steve). large(jim).
large(mike).
large_animal(X):- dog(X),large(X).
large_animal(Z):- cat(Z),large(Z).
```

fido, mary, jane etc. are atoms, i.e. constants, indicated by their initial lower case letters.
X and Y are variables, indicated by their initial capital letters.
The first 18 clauses are facts.
The final two clauses are rules.

2. Predicates

- All the clauses (facts and rules) for which the head has a given combination of functor and arity comprise a definition of a *predicate*.

Example:

```
parent(ali).
parent(ahmad,khaled).
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
father(ahmad,khaled).
mother(maha,khaled).
```

It is possible for the program to include clauses for which the heads have the same functors, but a different arity (no. of arguments).

2. Predicates (cont.)

- The clauses do not have to appear as consecutive lines of a program but it makes programs easier to read if they do.
- in textbooks, reference manuals etc., not inside programs, the predicates are written as **parent/2** and **parent/1**, to distinguish between them.
- A user's program comprises facts and rules that define new predicates. These are called *user-defined predicates.*

2. Predicates (cont.)

Declarative and Procedural Interpretations of Rules:

- Rules have both a *declarative and a procedural interpretation. For example, the* declarative interpretation of the rule:

```
chases(X,Y):-dog(X),cat(Y),write(X),
    write(' chases '),write(Y),nl.
```

- is: '**chases(X,Y) is true if dog(X) is true and cat(Y) is true and write(X) is true,** etc.'

2. Predicates (cont.)

Declarative and Procedural Interpretations of Rules:

- The procedural interpretation is 'To satisfy **chases(X,Y), first satisfy dog(X)**, then satisfy **cat(Y), then satisfy write(X), etc.**'
- The order of the clauses defining a predicate and the order of the goals in the body of each rule are irrelevant to the declarative interpretation but of vital importance to the procedural interpretation. The goals in the body of a rule are examined from left to right.

2. Predicates (cont.)

Simplifying Entry of Goals:

- In developing or testing programs it can be tedious to enter repeatedly at the system prompt a lengthy sequence of goals such as

  ?-dog(X),large(X),write(X),write(' is a large dog'),nl.

- A commonly used programming technique is to define a predicate such as **go/0** or **start/0,** with the above sequence of goals as the right-hand side of a rule,

2. Predicates (cont.)

Simplifying Entry of Goals:

Example:

```
go:-dog(X),large(X),write(X),
        write(' is a large dog'),nl.
```

- This enables goals entered at the prompt to be kept brief, e.g.
- ?-go.

2. Predicates (cont.)

- **Recursion:**
- An important technique for defining predicates, is to define them in terms of themselves. This is known as a *recursive definition*.
- *Example*:

  likes(ali,X):-likes(X,Y),cat(Y).

  which can be interpreted as 'Ali likes anyone who likes at least one cat'.

4. Variables

Variables in Goals:

- Variables in goals can be interpreted as meaning 'find values of the variables that make the goal satisfied'.
- **Example**: (This example is addition to those clauses in Slide#8):

```
chases(X,Y):- dog(X),cat(Y),
                write(X),write(' chases '),write(Y),nl.
/* chases is a predicate with two arguments*/
go:-chases(A,B).
```

4. Variables (cont.)

Variables in Goals:

- A goal such as

  ?-chases(X,Y).

- means find values of variables *X and Y to satisfy* ***chases(X,Y).***
- To do this, Prolog searches through all the clauses defining the predicate **chases** (there is only one in this case) from top to bottom until a matching clause is found.
- It then works through the goals in the body of that clause one by one, working from left to right, attempting to satisfy each one in turn.

4. Variables (cont.)

Binding Variables:

- Initially all variables used in a clause are said to be ***unbound***, *meaning that they do* not have values.
- When the Prolog system evaluates a goal some variables may be given values. *This is known as* ***binding the variables***.
- A variable that has been bound may become unbound again and possibly then bound to a different value by the process of *backtracking*.

4. Variables (cont.)

Universally Quantified Variables:

- If a variable appears in the head of a rule or fact it is taken to indicate that the rule or fact *applies for all possible values of the variable*.
- *For example, the rule*

```
large_animal(X):-dog(X),large(X).
```

- can be read as 'for all values of X, X is a large animal if X is a dog and X is large'.
- Variable X is said to be **universally quantified**

4. Variables (cont.)

Existentially Quantified Variables:

- Suppose now that the database contains the following clauses:

```
person(frances,wilson,female,28,architect).
person(fred,jones,male,62,doctor).
person(paul,smith,male,45,engineer).
person(martin,williams,male,23,chemist).
person(mary,jones,female,24,programmer).
person(martin,johnson,male,47,teacher).
man(A):-person(A,B,male,C,D).
```

- The last clause is a rule, defined using the person predicate, which also has a natural interpretation, i.e. 'for all *A*, *A is a man if A is a person whose sex is male*'.

4. Variables (cont.)

Existentially Quantified Variables:

- More helpful interpretation would be to take variable ***B*** *to mean* 'for at least one value of *B' and similarly for variables* ***C*** *and* ***D***.
- The key distinction between variable A and variables B, C and D in the definition of predicate man is that B, C and D do not appear in the head of the clause.
- The convention used by Prolog is that if a variable, say *Y, appears in the body* of a clause but not in its head it is taken to mean 'there is (or there exists) at least one value of Y'. Such variables are said to be ***existentially quantified***. *Thus the rule*

```
houseowner(X):-house(Y),owns(X,Y).
```

- can be interpreted as meaning 'for all values of X, X is a house owner if there is some Y such that Y is a house and X owns Y'.

4. Variables (cont.)

The Anonymous Variable:

- The underscore character _ denotes a special variable, **called the *anonymous variable***. *This is used when the user does not care about the value of the variable.*
- ***Example***: *to check whether there is someone with name paul in the database, an easier way is to use the goal:*

```
?- person(paul,_,_,_,_).
true.
```

- ***Example***: If only the surname of any people named *paul is of interest, this can be found*
- by making the other three variables anonymous in a goal, e.g.

```
?- person(paul,Surname,_,_,_).
Surname = smith
```

4. Variables (cont.)

The Anonymous Variable:

- Similarly, if only the ages of all the people named *martin in the database are of* interest, it would be simplest to enter the goal:

  ?- person(martin,_,_,Age,_).

- This will give two answers by backtracking.

  Age = 23 ;

  Age = 47

- The three anonymous variables are not bound, i.e. given values,

4. Variables (cont.)

The Anonymous Variable:

- Note that there is no assumption that all the anonymous variables have the same value (in the above examples they do not). Entering the alternative goal

  ?- person(martin,X,X,Age,X).

- with variable *X instead of underscore each time, would produce the answer*

  false

- as there are no clauses with first argument *martin where the second, third and fifth* arguments are identical.