Facts, rules, goals and queries

A Prolog program consists of a number of clauses. Each clause is either a fact or a rule. After a Prolog program is loaded (or *consulted*) in a Prolog interpreter, users can submit goals or queries, and the Prolog intepreter will give results (answers) according to the facts and rules.

Facts

A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be atoms (in this case, these atoms are treated as constants), numbers, variables or lists. Arguments are separated by commas.

If we consider the arguments in a fact to be objects, then the predicate of the fact describes a property of the objects.

In a Prolog program, a presence of a fact indicates a statement that is true. An absence of a fact indicates a statement that is not true. See the following example:

Program 2: A Prolog program with facts only

```
sunny.
father(john, peter).
father(john, mary).
mother(susan, peter).
```

Goals/Queries and their results (in red):

```
?- sunny.    /* The response is yes because the fact "sunny." is present. */

yes
```

```
?- rainy.    /* There is an error because there is no predicate "rainy". */
```

Erroneous result.

```
?- father(john, mary).

yes
```

```
?- mother(susan, mary). /* This cannot be deduced. */

no
```

```
?- father(john, susan). /* This cannot be deduced. */

no
```

Rules

A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true. It consists of two parts. The first part is similar to a fact (a predicate with arguments). The second part consists of other clauses (facts or rules which are separated by commas) which must all be true for the rule itself to be true. These two parts are separated by "`:-`". You may interpret this operator as "if" in English.

See the following example:

Program 3: A program describes the relationships of the members in a family

```
father(jack, susan).                              /* Fact  1 */
father(jack, ray).                                /* Fact  2 */
father(david, liza).                              /* Fact  3 */
father(david, john).                              /* Fact  4 */
father(john, peter).                              /* Fact  5 */
father(john, mary).                               /* Fact  6 */
mother(karen, susan).                             /* Fact  7 */
mother(karen, ray).                               /* Fact  8 */
mother(amy, liza).                                /* Fact  9 */
mother(amy, john).                                /* Fact 10 */
mother(susan, peter).                             /* Fact 11 */
mother(susan, mary).                              /* Fact 12 */

parent(X, Y) :- father(X, Y).                     /* Rule  1 */
parent(X, Y) :- mother(X, Y).                     /* Rule  2 */
grandfather(X, Y) :- father(X, Z), parent(Z, Y).  /* Rule  3 */
grandmother(X, Y) :- mother(X, Z), parent(Z, Y).  /* Rule  4 */
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).  /* Rule  5 */
yeye(X, Y) :- father(X, Z), father(Z, Y).         /* Rule  6 */
mama(X, Y) :- mother(X, Z), father(Z, Y).         /* Rule  7 */
gunggung(X, Y) :- father(X, Z), mother(Z, Y).     /* Rule  8 */
popo(X, Y) :- mother(X, Z), mother(Z, Y).         /* Rule  9 */
```

Take Rule 3 as an example. It means that "`granfather(X, Y)`" is true if both "`father(X, Z)`" and "`parent(Z, X)`" are true. The comma between the two conditions can be considered as a logical-AND operator.

You may see that both Rules 1 and 2 start with "`parent(X, Y)`". When will "`parent(X, Y)`" be true? The answer is any one of these two rules is true. This means that "`parent(X, Y)`" is true when "`father(X, Y)`" is true, or "`mother(X, Y)`" is true.

As you can see from Rules 3 to 5, predicates that only appear in rules but not facts (in this case, `parent`) can also form conditions of other rules.

Goals and queries

When a Prolog interpreter is running, you may probably see the prompt "`?-`" on the screen. This prompts the user to enter a goal or a query.

Goals

A goal is a statement starting with a predicate and probably followed by its arguments. In a valid goal, the predicate must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the goal must be the same as that appears in the consulted program. Also, all the arguemnts (if any) are constants.

The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database (i.e. the facts and rules in the consulted program). This is similar to proving a hypothesis - the goal being the hypothesis, the facts being the axioms and the rules being the theorems.

Queries

A query is a statement starting with a predicate and followed by its arguments, some of which are variables. Similar to goals, the predicate of a valid query must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the query must be the same as that appears in the consulted program.

The purpose of submitting a query is to find values to substitute into the variables in the query such that the query is satisfied. This is similar to asking a question, asking for "what values will make my statement true".

The following examples show how goals and queries are evaluated. The program used is Program 3. Results are shown in red.

`?- parent(susan, mary).`
 	
`yes`
	This is a goal proving that "Susan is a parent of Mary". From Fact 12 and Rule 2, we can see that it is true.

`?- parent(ray, peter).`
 	
`no`
	This is a goal proving that "Ray is a parent of Peter". Since no facts and rules support it, this statement is disproved.

`?- yeye(X, susan).`
 	
`no`
	This is a query asking for the person who is the "yeye" of Susan. We cannot find the solution from the program, so the Prolog interpreter returns `no`.

`?- mama(amy, X).`
 	
`X = peter ;`
 	
`X = mary ;`
 	
`no`
	This is a query asking for the person who calls Amy "mama". From the program, we can see that both Peter and Mary are solutions. Therefore the Prolog interpreter display both answers.
	
	Since the Prolog interpreter can only display one solution at a time, a semicolon (this is software dependent) is entered to ask for the next solution. If there is no more solution, `no` will be returned.

`?- gunggung(X, Y).`
 	
`X = jack`
`Y = peter ;`
 	
`X = jack`
`Y = mary ;`
 	
`no`
	This is a query asking for: "X is a 'gunggung' of Y'. Who are X and Y?" From the program, we can see that X is Jack, while Y can be Peter or Mary. Therefore the Prolog interpreter displays both set of results.

See the next page to see how the Prolog interpreter deals with goals and queries.

Common built-in predicates

In addition to self-defined predicates, Prolog also provides built-in predicates. The following are some common built-in predicates:

- Arithmetic predicates (arithmetic operators): `+`, `-`, `*`, `/` (These operators are the same as those found in other programming languages. They only operates on numbers and variables.)
- Comparison predicates (comparison operators):
  - `<` (less than) - operates on numbers and variables only
  - `>` (greater than) - operates on numbers and variables only
  - `=<` (less than or equal to) - operates on numbers and variables only
  - `>=` (greater than or equal to) - operates on numbers and variables only
  - `is` - the two operands have the same values
  - `=` - the two operands are identical
  - `=\=` - the two operands do not have the same values

Note that each of the built-in predicates above have two arguments, one on the left of the predicate and one on the right (just similar to other programming languages). However, user-defined predicates (as well as some other built-in predicates) must come before their arguments.

Also note the difference between `is` and `=` . See the following goals:

`?- 4 = 4.` `yes`	Obvious.
`?- 4 is 4.` `yes`	Obvious.
`?- 4 = 1 + 3.` `no`	The answer is `no` because the left hand side (`4`) is not identical to right hand side (`1 + 3`).
`?- 4 is 1 + 3.` `yes`	The value of left hand side is equal to the value of right hand side.