Facts, Rules and Queries

Symbols

Prolog expressions are comprised of the following truth-functional symbols, which have the same interpretation as in the predicate calculus.

English	Predicate Calculus	PROLOG
and	^	,
or	v	;
if	-->	:-
not	~	not

Variables and Names

Variables begin with an uppercase letter. Predicate names, function names, and the names for objects must begin with a lowercase letter. Rules for forming names are the same as for the predicate calculus.

```
mother_of
male
female
greater_than
socrates
```

Facts

A **fact** is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be **universally quantified**. Note that all Prolog sentences must end with a period.

```
likes(john, susie).                    /* John likes Susie */
likes(X, susie).                       /* Everyone likes Susie */
likes(john, Y).                        /* John likes everybody */
likes(john, Y), likes(Y, john).        /* John likes everybody and everybody likes John */
likes(john, susie); likes(john,mary). /* John likes Susie or John likes Mary */
not(likes(john,pizza)).                /* John does not like pizza */
likes(john,susie) :- likes(john,mary)./* John likes Susie if John likes Mary.
```

Rules

A **rule** is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus a Prolog rule takes the form

```
left_hand_side :- right_hand_side .
```

This sentence is interpreted as: *left_hand_side **if** right_hand_side*. The **left_hand_side** is restricted to a **single, positive, literal**, which means it must consist of a positive atomic expression. It cannot be negated and it cannot contain logical connectives.

This notation is known as a **Horn clause**. In Horn clause logic, the left hand side of the clause is the conclusion, and must be a single positive literal. The right hand side contains the premises. The Horn clause calculus is equivalent to the first-order predicate calculus.

Examples of valid rules:

```
friends(X,Y) :- likes(X,Y),likes(Y,X).             /* X and Y are friends if they like each other */
hates(X,Y) :- not(likes(X,Y)).                     /* X hates Y if X does not like Y. */
enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)).   /* X and Y are enemies if they don't like each other */
```

Examples of invalid rules:

```
left_of(X,Y) :- right_of(Y,X)                      /* Missing a period */
likes(X,Y),likes(Y,X) :- friends(X,Y).             /* LHS is not a single literal */
not(likes(X,Y)) :- hates(X,Y).                     /* LHS cannot be negated */
```

Queries

The Prolog interpreter responds to **queries** about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" and displays any **variable bindings** that it made in coming up with the answer. If it fails to prove the query true, it answers "No".

Whenever you run the Prolog interpreter, it will **prompt** you with **?-**. For example, suppose our database consists of the following facts about a fictitious family.

```
father_of(joe,paul).
father_of(joe,mary).
mother_of(jane,paul).
mother_of(jane,mary).
male(paul).
male(joe).
female(mary).
female(jane).
```

We get the following results when we make queries about this database. (I've added comments, enclosed in /*..*/, to interpret each query.)

```
Script started on Wed Oct 01 14:29:32 2003
sh-2.05b$ gprolog
GNU Prolog 1.2.16
By Daniel Diaz
Copyright (C) 1999-2002 Daniel Diaz
| ?- ['family.pl'].
compiling /home/ram/public_html/cpsc352/prolog/family.pl for byte code...
/home/ram/public_html/cpsc352/prolog/family.pl compiled, 9 lines read - 999 bytes written, 94 ms

(10 ms) yes
| ?- listing.

mother_of(jane, paul).
mother_of(jane, mary).

male(paul).
male(joe).

female(mary).
female(jane).

father_of(joe, paul).
father_of(joe, mary).

(10 ms) yes
| ?- father_of(joe,paul).

true ?
yes
| ?- father_of(paul,mary).

no
| ?- father_of(X,mary).

X = joe

yes
| ?-
Prolog interruption (h for help) ? h
   a  abort        b  break
   c  continue     e  exit
   d  debug        t  trace
  h/? help

Prolog interruption (h for help) ? e
sh-2.05b$ exit

script done on Wed Oct 01 14:30:50 2003
```

Closed World Assumption. The Prolog interpreter assumes that the database is a **closed world** -- that is, if it cannot prove something is true, it assumes that it is false. This is also known as **negation as failure** -- that is, something is false if PROLOG cannot prove it true given the facts and rules in its database. In this case, in may well be (in the real world), that Paul is the father of Mary, but since this cannot be proved given the current family database, Prolog concludes that it is false. So PROLOG assumes that its database contains complete knowledge of the domain it is being asked about.

Prolog's Proof Procedure

In responding to queries, the Prolog interpreter uses a **backtracking** search, similar to the one we study in Chapter 3 of Luger. To see how this works, let's add the following rules to our database:

```
parent_of(X,Y) :- father_of(X,Y).       /* Rule #1 */
parent_of(X,Y) :- mother_of(X,Y).       /* Rule #2 */
```

And let's trace how PROLOG would process the query. Suppose the facts and rules of this database are arranged in the order in which they were input. This trace assumes you know how unification works.

```
?- parent_of(jane,mary).

parent_of(jane,mary)     /* Prolog starts here and searches
                            for a matching fact or rule. */
  parent_of(X,Y)         /* Prolog unifies the query with the rule #1
                            using {jane/X, mary/Y}, giving
                            parent_of(jane,mary) :- father_of(jane,mary) */
  father_of(jane,mary)   /* Prolog replaces LHS with RHS and searches. */
                         /* This fails to match father_of(joe,paul) and
                            and father_of(joe,mary), so this FAILS. */
                         /* Prolog BACKTRACKS to the other rule #2 and
                            unifies with {jane/X, mary/Y}, so it matches
                            parent_of(jane,mary) :- mother_of(jane,mary) */
  mother_of(jane,mary)   /* Prolog replaces LHS with RHS and searches. */

  YES.                   /* Prolog finds a match with a literal and so succeeds.
```

Here's a trace of this query using Prolog's **trace** predicate:

```
| ?- trace,parent_of(jane,mary).
{The debugger will first creep -- showing everything (trace)}
   1  1  Call: parent_of(jane,mary) ?
   2  2  Call: father_of(jane,mary) ?
   2  2  Fail: father_of(jane,mary) ?
```

```
   2  2  Call: mother_of(jane,mary) ?
   2  2  Exit: mother_of(jane,mary) ?
   1  1  Exit: parent_of(jane,mary) ?

yes
{trace}
| ?-
```

Exercises

1. Download the family.pl file.
2. Add a `male()` rule that includes all fathers as males.
3. Add a `female()` rule that includes all mothers as females.
4. Add the following rules to the family database:

   ```
   son_of(X,Y)
   daughter_of(X,Y)
   sibling_of(X,Y)
   brother_of(X,Y)
   sister_of(X,Y)
   ```

5. Given the addition of the `sibling_of` rule, and assuming the above order for the facts and rules, show the PROLOG trace for the query `sibling_of(paul,mary)`.