### 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).

%query knowledge base

CHECK THE STATEMENT

PRIYA CAN COOK FOOD

girl(priya).

girl(tiyasha).

girl(jaya).

can\_cook(priya).

Output:-

GNU Prolog 1.4.4 (32 bits)

Compiled May 15 2013, 12:35:56 with i686-w64-mingw32-gcc

By Daniel Diaz

Copyright (C) 1999-2013 Daniel Diaz

compiling C:/GNU-Prolog/pro/kb1.pl for byte code...

C:/GNU-Prolog/pro/kb1.pl compiled, 3 lines read - 473 bytes written, 0 ms

| ?- change\_directory('C:/GNU-Prolog/pro').

yes

| ?- [kb1].

compiling C:/GNU-Prolog/pro/kb1.pl for byte code...

C:/GNU-Prolog/pro/kb1.pl compiled, 4 lines read - 473 bytes written, 10 ms

yes

| ?- girl(priya).

yes

| ?- girl(tanveer).

no

| ?-can\_cook(reshma).

no

| ?- can\_cook(priya).

Yes

%check with fact

* Tom is a cat
* Kunal loves to eat Pasta
* Hair is black
* Nawaz loves to play games
* Pratyusha is lazy.

PROGRAM

cat(tom).

loves\_to\_eat(kunal,pasta).

of\_color(hair,black).

loves\_to\_play\_games(nawaz).

lazy(pratyusha).

Rules

We can define rule as an implicit relationship between objects. So facts are conditionally true. So when one associated condition is true, then the predicate is also true. Suppose we have some rules as given below −

* Lili is happy if she dances.
* Tom is hungry if he is searching for food.
* Jack and Bili are friends if both of them love to play cricket.
* will go to play if school is closed, and he is free.

Program

happy(lili) :- dances(lili).

hungry(tom) :- search\_for\_food(tom).

friends(jack, bili) :- lovesCricket(jack), lovesCricket(bili).

goToPlay(ryan) :- isClosed(school), free(ryan).

%check with query

RAM LIKES THE MANGO

BILL LIKES CINDY

JOHN OWNS GOLD

A ROSE IS RED COLOUR

PROGRAM

likes(ram ,mango).

girl(seema).

red(rose).

likes(bill ,cindy).

owns(john ,gold).

%JOHN THE THIEF STATEMENTS

A THIEF WINI HE LIKE A BALL

A THIEF WILLIAM HE LIKE A SHOE

A THIEF JOHN LIKE SNOW AND DOLLAR

FIND THE STEAL PERSON AND LIKES

PROGRAM

thief(wini).

thief(john).

thief(willisam).

likes(wini,ball).

likes(william,shoe).

likes(john,snow).

likes(john,dollar).

may\_steal(john,X) :-thief(john), likes(john,X).

output:-

| ?- [kb3].

compiling C:/GNU-Prolog/pro/kb3.pl for byte code...

C:/GNU-Prolog/pro/kb3.pl compiled, 10 lines read - 1088 bytes written, 12 ms

yes

| ?- likes(john,X).

X = snow ? ;

X = dollar

yes

| ?- may\_steal(john,X).

X = snow ? ;

X = dollar

yes

| ?-

may\_steal(john,ball).

no

| ?- may\_steal(john,dollar).

Yes

%fact and query knowledge base STATEMENTS

A DOG NAME IS SAM

A DOG NAME IS HENRY

A CAT NAME IS FELIX

A CAT NAME IS MICHAEL

FIND DOG AND CAT IN THIS QUERY

PROGRAM

dog(sam).

dog(henry).

cat(felix).

cat(jame).

cat(michael).

animal(X):-dog(X).

animal(Y):-cat(Y).

output:-

?- [kb4].

compiling C:/GNU-Prolog/pro/kb4.pl for byte code...

C:/GNU-Prolog/pro/kb4.pl compiled, 7 lines read - 685 bytes written, 0 ms

yes

| ?- animal(X).

X = sam ? ;

X = henry ? ;

X = felix ? ;

X = jame ?

yes

| ?- animal(Y).

Y = sam ? ;

Y = henry ? ;

Y = felix ? ;

Y = jame ? ;

Y = michael

(31 ms) yes

| ?- dog(X).

X = sam ? ;

X = henry

(16 ms) yes

| ?- cat(Y).

Y = felix ? ;

Y = jame ? ;

Y = michael

yes

| ?-

%PROGRAM OF ARITHMATIC OPERATION

sum(X,Y):-

S is X+Y,

write(S).

sub(X,Y):-

S is X-Y,

write(S).

mul(X,Y):-

M is X\*Y,

write(M).

div(X,Y):-

D is X/Y,

write(D).

output:-

| ?- sum(10,5).

15

yes

| ?- sub(10,5).

5

(15 ms) yes

| ?- mul(20,5).

100

yes

| ?- div(100,5).

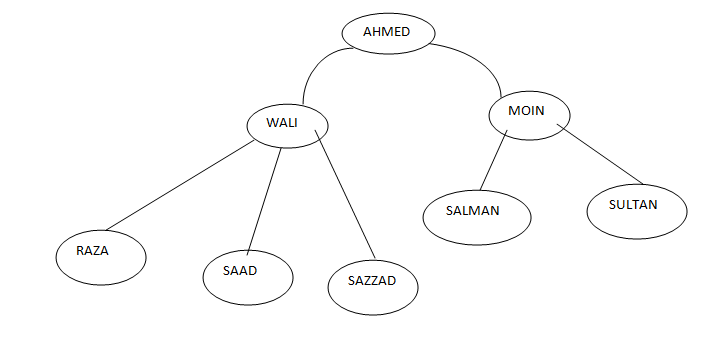
20.0

% FAMILY TREE

% CHECK GRANDFATHER RELATIONSHIP

%CHECK UNCLE RELATIONSHIP

%CHECK COUSIN RELATIONSHIP



father(ahmed,wali).

father(ahmed,moin).

father(wali,raza).

father(wali,saad).

father(wali,sazzad).

father(moin,salman).

father(moin,sultan).

sibling(wali,moin).

sibling(moin,wali).

sibling(raza,saad).

sibling(saad,raza).

sibling(raza,sazzad).

sibling(sazzad,raza).

sibling(saad,sazzad).

sibling(sazzad,saad).

sibling(salman,sultan).

sibling(sultan,salman).

grandfather(X,Y) :-

father(X,Z) , father(Z,Y).

uncle(A,B) :-

father(X,A) , father(X,Y) , father(Y,B) , sibling( A,Y).

cousin(X,Y) :-

father(M,X) , father(N,Y) , sibling(M,N).

output:-

compiling C:/GNU-Prolog/pro/kb6.pl for byte code...

C:/GNU-Prolog/pro/kb6.pl compiled, 30 lines read - 2092 bytes written, 10 ms

yes

| ?- grandfather(wali,raza).

no

| ?- grandfather(ahmed,raza).

true ?

yes

compiling C:/GNU-Prolog/pro/kb6.pl for byte code...

C:/GNU-Prolog/pro/kb6.pl compiled, 30 lines read - 2616 bytes written, 10 ms

yes

| ?- uncle(wali,sultan).

true ?

(15 ms) yes

| ?- uncle(moin,raza).

true ?

yes

| ?- uncle(moin,salman).

no

| ?-

[kb6].

compiling C:/GNU-Prolog/pro/kb6.pl for byte code...

C:/GNU-Prolog/pro/kb6.pl compiled, 30 lines read - 3038 bytes written, 10 ms

(31 ms) yes

| ?- cousin(raza,sultan).

true ?

yes

| ?- cousin(moin,raza).

no

| ?-