1 Prolog

1.1 Introduction

Prolog is a logic programming language.

A Prolog program is logic—a collection of facts and rules for proving things. Prolog programs are not "run"; you pose questions and the language system uses the program's collection of facts and rules to try to answer them.

You can do anything in Prolog that you can in any other general purpose programming language. (It may not be easy.) Prolog is especially useful in domains that involve searching for solutions to problems that are specified logically. Prolog is one of the two most popular languages for AI programming (Lisp is the other).

1.2 Prolog Terms

Everything in a Prolog program—both the program and the data it manipulates—is build from Prolog terms.

1. Constants
2. Variables
3. Compound terms (or structures)

1.2.1 Constants

Integers, real numbers, or an *atom*.

Any name that starts with a lowercase letter (followed by zero or more additional letters, digits, or underscores) is an *atom*. Atoms look like variables of other languages, but are treated as constants in Prolog. The atom `n` is never equal to anything but the atom `n`.

Sequences of most non-alphanumeric characters (+, *, -, etc.) are also atoms.

Special atoms:
`[]` empty list
`!` *cut* and
`;` disjunction

1.2.2 Variables

A variable is any name beginning with an uppercase later or an underscore, followed by zero or more additional letters, digits, or underscores.

```
X
Y
```

_ is the anonymous variable.

1.2.3 Compound terms

Compound terms have an atom followed by a parenthesized, comma-separated list of terms. For example:

```
parent(mark,john).
parent(mark,Child).
```

Note Compound terms may look like function calls in other languages, but they almost never work anything like function calls. It is best to think of them as structured data

1.2.4 Unification

Pattern-matching using Prolog terms is called unification. Prolog makes extensive use of pattern matching. Two terms are said to *unify* if there is some way of binding their variables that makes them identical. Consider these two terms:

```
parent(mark,john).
parent(mark,Child).
```

These unify by binding the variable `Child` to the atom `john`. Finding a way to unify two terms can be tricky.

1.2.5 Facts

Define facts

```
parent(mark,john).
```

- The names of all relationship and objects must begin with a lowercase letter.
- The relationship is written first, and the objects are written separated by commas, and the objects are enclosed by a pair or parentheses.
- A period ('.') must come at the end of a fact.

Note: Must be careful about the order of the objects in the relationship! Examples

```
parent(mark,john).   ;;  mark is the parent of john
parent(sally,sue).
parent(sally,john).
parent(mary,sally).  ;;  mary is sue's grandmother.  order?
parent(steve,mark).
```

An atom that starts a compound term with n parameters is called a *predicate of arity n.*

The program above gives some facts about a `parent` predicate of arity 2.

1.3 Using a Prolog System

SWI-Prolog should be on the Windows systems. Exit `halt.`

```
Welcome to SWI-Prolog (Version 5.0.10)
Copyright (c) 1990-2002 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- consult(relations).     . at the end of a query
```

`consult` predicate adds the contents of a file to the system's internal data base.

```
?- parent(mary,sally).
Yes
?- parent(barney,bambam).
No
?-

?- parent(P,sally).
P = mary

Yes
?- parent(P,pebbles).
No
?- parent(mary,Child).
Child = sally
Yes
?- parent(Parent,Child)
Parent = mary
Child = sally
```

1.3.1 Conjunctions

```
likes(john,food).
likes(john,coke).
likes(john,motorcycles).    Order?
likes(jack,food).
likes(jack,pepsi).
likes(jack,football).
```

Do John and Jack like food?

```
?-  likes(john,food), likes(jack,food).
Yes

?-  likes(john,motorcycles), likes(jack,motorcycles)
No
```

The comma is pronounced "and", and it serves to separate any number of different goals

Is there anything that both Jack and John both like?

```
?- likes(john,X), likes(jack,X)
```

Prolog answers the question by attempting to satisfy the first goal. If the first goal is in the database, then Prolog marks the place in the database and attempts to satisfy the second goal.

Each goal keeps its own place!

1.3.2 Rules

Rules are used when you want to say a fact *depends* on a group of other facts. A rule is a general statement about objects and their relationships.

Rules consist of a *head* and a *body*. The head and body are connected by the symbol `:-` (colon and a hyphen) and is pronounced *if*.

```
male(john).       facts
male(john).

female(mary).
female(sally).

parents(john,adam,sally).
parents(mary,adam,sally)

sister_of(X,Y) :=
   female(X),
   parents(X,M,F),
   parents(Y,M,F).     M and F indicate mother and father
```

1.3.3 Structures (compound terms) Again

Recall that compound terms (structures) are a collection of other objects, called components.

Structures help to organize the data in a program because they permit a group of related information to be treated as a single object instead of separate entities.

Structures are written by specifying its *functor*, and its *components*. The functor names the general kind of structure, and corresponds to a datatype in many programming languages.

Structures can be used in the question-answering process by using variables.

```
owns(bruce,book(prolog,clocksin_mellish)).
owns(bruce,book(the_c_Programming_Language,kernighan)).

owns(bruce,book(X,clocksin_mellish)).

owns(bruce,book(X,_)).   ??
```

Note the syntax for structures is the same as for facts. A predicate is actually the functor of a structure. The arguments of a fact or rule are actually the components of a structure.

_ is the anonymous variable.

1.3.4 Equality and Matching

= is an infix operator that checks to see if there is a match and is pronounced "equals".

```
?- X = Y.
```

\= is a predicate pronounced "not equal".

```
?- X \= Y.
```

1.3.5 Arithmetic

Typical relational operators

```
X = Y   X and Y stand for the same number
X \= Y  X and Y stand for different numbers
X < Y   X is less than Y
X > Y   X is greater than Y
X =< Y  X is less than or equal to Y
X >= Y  X is greater than or equal to Y
```

1.3.6 Population density

```
pop(usa,280).       /* 280 million  */
pop(india,1000).    /*   1 billion  */
pop(china,1200).    /* 1.2 billion  */
pop(brazil,130).

area(usa,3).        /* millions of square miles */
area(india,1).
area(china,4).
area(brazil,3).

density(X,Y) :-     The population density of country X is Y, if:
  pop(X,P),            The population of X is P, and
  area(X,A),           The area of X is A, and
  Y is P/A.            Y is calculated by dividing P by A.

- consult(population).
% population compiled 0.00 sec, 1,548 bytes
Yes

?- density(usa,D).
D = 93.3333
Yes

?- density(china,D).
D = 300
Yes
```

Note the use of the `is` operator. Prolog evaluates its righthand argument according to the rules of arithmetic. `is` is required to evaluate arithmetic expressions.

1.3.7 Factorial

```
/*  factorial.pl  */

factorial(0,1).
factorial(N,F) :-
   N > 0,
   N1 is N-1,
   factorial(N1,F1),
   F is N * F1.

?- consult(factorial).
% factorial compiled 0.00 sec, 736 bytes
Yes
?- factorial(3).
ERROR: Undefined procedure: factorial/1
ERROR:     However, there are definitions for:
ERROR:         factorial/2

No

?- factorial(3,F).
F = 6
Yes
```

1.3.8 Farmer, Wolf, Goat, and Cabbage Crossing

Wolf eats goat if farmer not there. Goat eats (very large) Cabbage if farmer not there. Boat can only hold two things at a time.

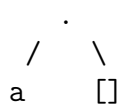
```
change(e,w).
change(w,e).

move([X,X,Goat,Cabbage],wolf,[Y,Y,Goat,Cabbage]) :-
   change(X,Y).
move([X,Wolf,X,Cabbage],goat,[Y,Wolf,Y,Cabbage]) :-
   change(X,Y).
move([X,Wolf,Goat,X],cabbage,[Y,Wolf,Goat,Y]) :-
   change(X,Y).
move([X,Wolf,Goat,C],nothing,[Y,Wolf,Goat,C]) :-
   change(X,Y).

oneEq(X,X,_).
oneEq(X,_,X).

safe([Man,Wolf,Goat,Cabbage]) :-
   oneEq(Man,Goat,Wolf),
   oneEq(Man,Goat,Cabbage).

solution([e,e,e,e],[]).
solution(Config,[Move|Rest]) :-
   move(Config,Move,NextConfig),
   safe(NextConfig),
   solution(NextConfig,Rest).
```

1.4 Lists in Prolog

Lists are ordered sequences of elements that can have any length.

Lists can be represented as a special kind of tree. A list is either empty, or it is a structure that has two components: the head and tail.

List of one element is `.(a.[])`



List notation consists of the elements of the list separated by commas, and the whole list is enclosed in square brackets. For example, `[a]` and `[a,b,c]`. Lists can contain other lists.

Split a list into its head and tail using the operation `[X|Y]`. For example:

```
p([1,2,3]).
p([the,cat,sat,[on,the,hat]]).

?- p([X|Y]).
X = 1    Y = [2,3] ;
X = the  Y = [cat,sat,[on,the,hat]]
```

1.4.1 List Membership

```
member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).
```