Logic Programming in Prolog
Part I
Basics

Raj Sunderraman

Prolog Syntax

Terms

Terms represent data objects. There are 3 types of terms:

- Atoms:
  - symbolic atoms begin with lower-case letter, e.g. tom, bill, a1, …
  - numeric atoms, e.g. 217, -32, 2.76, …
  - strings, e.g. “hello”, “tony”,…
- Variables:
  begin with upper-case letter or underscore,
  e.g. X, U, _x1, Tom, A1
  _ represents anonymous variable
- Structures:
  - function structure: f(t1,…,tn), n>=0
  - list structure: [t1,…,tn], n>=0

  where f is a symbolic atom and t1,…,tn are terms, e.g.
  edge(3,7), f(g(1),h(2,3)), f(x), …
  [a,b,c], [1,2,3,4], [f(a),X,2.76,Z,z], [[a],[b]], …

  Special notation for lists: [H|T], [X,Y|T], [c1,…,cn|T]

Prolog Syntax Continued

Relation

ordinary relation: r(t1,…,tn), n>=0
where r is a symbolic atom and t1,…,tn are terms, e.g.
round, father(tom,bill), student(1111,jones,freshman,4.0), ancestor(X,Y),
groceryList(monday,[[“Apples”,12],[“Bananas,10]]),…

built-in relation: written in infix notation usually, e.g.
X \== Y, X > 20, …

Program
consists of a finite number of facts and rules;

fact: relation.
rule: relation :- relation-1,…,relation-n.
n>=1

e.g.
```
member(X,[X|_]).
member(X,[Y|T]) :- X \==Y, member(X,T).
```

Prolog Syntax Continued

Query

?- relation-1,…,relation-n.
used to run a program

SWI Prolog http://www.swi-prolog.org/

```
[raj@tinman prologCode]$ more p1.pl
sizeOfList([],0).
sizeOfList([_|T], N) :- sizeOfList(T,M), N is M+1.

[raj@tinman prologCode]$ swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU 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).

?- ['p1.pl'].
true.

?- sizeOfList([1,2,3,4,5,6],N).
N = 6.

?- halt.
[raj@tinman prologCode]$
```

Sample Programs

```
parent(a,b).
parent(b,c).
parent(c,d).
parent(d,e).
parent(c,f).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

sibling(X,Y) :- parent(Xp,X), parent(Xp,Y), X \== Y.
cousin(X,Y) :- parent(Xp,X), parent(Yp,Y), sibling(Xp,Yp).
cousin(X,Y) :- parent(Xp,X), parent(Yp,Y), cousin(Xp,Yp).

grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

List Processing in Prolog

```
append([],L,L).
append([A|L],Y,[A|Z]) :- append(L,Y,Z).

prefix(P,L) :- append(P,_,L).

suffix(S,L) :- append(_,S,L).

reverse([],[]).
reverse([A|L],M) :- reverse(L,L1), append(L1,[A],M).

palindrome(L) :- reverse(L,L).

min([A],A).
min([A|L],A) :- min(L,N), A < N.
min([A|L],N) :- min(L,N), A >= N.

max([A],A).
max([A|L],A) :- max(L,N), A > N.
max([A|L],N) :- max(L,N), A =< N.
```

List Processing in Prolog
Tail Recursion

```
reverse(L,R) :- reverse1(L,[],R).

reverse1([],P,P).
reverse1([X|T],P,S) :- reverse1(T,[X|P],S).

min([A],A).
min([A|T],Min) :- min1(T,A,Min).

min1([],A,A).
min1([X|T],A,Min) :- X =< A, min1(T,X,Min).
min1([X|T],A,Min) :- min1(T,A,Min).

similarly for max.
```

More Prolog Programs

```
factorial(0,1).
factorial(N,Nfac) :-
  N > 0,
  M is N-1,
  factorial(M,Mfac),
  Nfac is N * Mfac.

odd(s(0)).
odd(s(s(X))) :- odd(X).
```

Sorting - merge sort

```
mergesort([],[]).
mergesort([X],[X]).
mergesort(L,M) :-
  msplit(L,L1,L2),
  mergesort(L1,M1),
  mergesort(L2,M2),
  merge(M1,M2,M).

msplit([],[],[]).
msplit([X],[X],[]).
msplit([X,Y|L],[X|L1],[Y|L2]) :- msplit(L,L1,L2).

merge([],L,L).
merge(L,[],L).
merge([X|L],[Y|M],[X|N]) :- X =< Y, merge(L,[Y|M],N).
merge([X|L],[Y|M],[Y|N]) :- X > Y, merge([X|L],M,N).
```

Sorting - quick sort

```
quicksort([],[]).
quicksort([H|T],S) :-
  qsplit(H,T,L1,L2),
  quicksort(L1,M1),
  quicksort(L2,M2),
  append(M1,[H|M2],S).

qsplit(_,[],[],[]).
qsplit(H,[A|L],[A|L1],L2) :- A =< H, qsplit(H,L,L1,L2).
qsplit(H,[A|L],L1,[A|L2]) :- A > H, qsplit(H,L,L1,L2).
```

subsets, permutations

```
del(X,[X|T],T).
del(X,[Y|T],[Y|S]) :- del(X,T,S).

subset([],[]).
subset(S,R) :- del(X,S,T), subset(T,R).
subset(S,[X|R]) :- del(X,S,T), subset(T,R).

permute([],[]).
permute(L,[X|P]) :- del(X,L,M), permute(M,P).
```

Given two equal sized list, generate all possible pairings.

```
generate(L1,L2,Pairs) :- permute(L2,M2), combine(L1,M2,Pairs).

combine([],[],[]).
combine([A|S],[B|T],[[A,B]|U]) :- combine(S,T,U).
```

Logic Puzzle

Three friends competed in a programming contest and came first, second, and third respectively. Each of them has a different first name, likes a different sport, and has a different nationality.

Write a Prolog Program to answer the questions:

Who is the Australian?
What sport did Richard play?

Some additional hints:

- Michael likes basketball and did better than the American.
- Simon, the Israeli, did better than the tennis player.
- The cricket player came first.

Logic Puzzle - continued

We will use the structure person(Name,Nationality,Sport) to represent the three persons and we will create a list of 3 such structures.

```
name(person(A,_,_),A).
nationality(person(_,N,_),N).
sport(person(_,_,S),S).

didBetter(A,B,[A,B,_]).
didBetter(A,C,[A,_,C]).
didBetter(B,C,[_,B,C]).

first(X,[X|_]).

makeListOfFriends(0,[]).
makeListOfFriends(N,[person(_,_,_)|L]) :-
  M is N-1,
  makeListOfFriends(M,L).
```

Logic Puzzle - continued

```
answer(Aussie,RichardSport) :-
  makeListOfFriends(3,Friends),

  didBetter(M1,M2,Friends), name(M1,michael),
  sport(M1,basketball), nationality(M2,american),

  didBetter(M3,M4,Friends), name(M3,simon),
  nationality(M3,israeli), sport(M4,tennis),

  first(M5,Friends), sport(M5,cricket),

  member(Q1,Friends), name(Q1,Aussie), nationality(Q1,australian),

  member(Q2,Friends), name(Q2,richard), sport(Q2,RichardSport).
```