Logic Programming with Prolog

Prolog programs are constructed from **terms:**

Constants can be either atoms or numbers:

- Atoms: Strings of characters starting with a lower-case letter or enclosed in apostrophes.
- Numbers: Strings of digits with or without a decimal point.

Variables are strings of characters beginning with an upper-case letter or an underscore.

Structures consist of a **functor** or **function symbol** (looks like an atom), followed by a list of terms inside parentheses, separated by commas.

Structures have two interpretations:

As **predicates** (relations):

presidentof(marySue, iowa).

prime(7).

between(rock,X,hardPlace).

As **structured objects** similar to records:

computer(name(herky), locn('MLH 303'), make('IBM'), model('RS6000'))

list(3, list(5, list(8, list(13, nil))))

Prolog Programs

A Prolog program is a sequence of statements, called **clauses**, of the form

P_0 :- P_1, P_2, …, P_n**.**

Each of P_0, P_1, P_2, …, P_n is an atom or structure.

A **period** terminates every Prolog clause.

Declarative meaning:

"P_0 is true if P_1 and P_2 and … and P_n are true"

Procedural meaning:

"To satisfy goal P_0, satisfy goal P_1 then P_2 then … then P_n".

- P_0 is called the **head** goal of a clause.
- Conjunction of goals P_1, P_2, …, P_n forms the **body** of the clause.
- A clause without a body is a **fact**:

  "P." means "P is true" or "goal P is satisfied"

- A clause without a head

  ":- P_1, P_2, …, P_n**.**" or "?- P_1, P_2, …, P_n**.**"

  is a **query** interpreted as

  "Are P_1 and P_2 and … and P_n true?" or "Satisfy goal P_1 then P_2 then … then P_n"

Lists in Prolog

A list of terms can be represented between brackets:

[a, b, c, d]

Its head is "a" and its tail is "[b, c, d]".

The tail of [a] is [ ], the empty list.

Lists may contain lists:

[3.3, [a, 8, [ ]], [x], [p,q]] is a list of four items.

Special form to direct pattern matching:

- The term [X|Y] matches any list with at least one element:

  X matches the head of the list, and

  Y matches the tail.

- The term [X,Y,77|T] matches any list with at least three elements whose third element is the number 77:

  X matches the first element,

  Y matches the second element, and

  T matches rest of the list after the third item.

Using these pattern matching facilities, values can be specified as the intersection of constraints on terms instead of by direct assignment.

Recursion

Most interesting algorithms involve repeating some group of actions.

Prolog implements repetition using recursion.

Recursion is closely related to mathematical induction, requiring two cases:

Basis: Solve some initial or small version of the problem directly.

Recursion: Assuming the algorithm works on smaller or simpler versions of the problem, solve an arbitrary instance of the problem.

Example

sublist(S,L) succeeds if and only if the list S is a sublist of the list L.

sublist([a,b,c], [a,b,c,d,e]) succeeds.

sublist([c,d], [a,b,c,d,e]) succeeds.

sublist([b,d], [a,b,c,d,e]) fails.

For list algorithms, the basis usually deals with an empty list, certainly the smallest list.

(Some algorithms for lists do not handle the empty list and so begin with a singleton list, [H]).

For the recursion step, we define the algorithm for the arbitrary list, [H|T], assuming that it works correctly for its tail T, a smaller list.

Sublist basis

The empty list is a sublist of any list.

```
sublist([ ], L).                                    % 1
```

Sublist recursion

The list [H|T] is a sublist of the list [H|U] if the list T is a sublist of the list U starting at the first position.

```
sublist([H|T], [H|U] :- initialsublist(T,U).        % 2
initialsublist([ ], L).                             % 3
initialsublist([H|T],[H|U] :- initialsublist(T,U). % 4
```

Or the list S is a sublist of the list [H|U] if it is a sublist of U.

```
sublist(S, [H|U] :- sublist(S,U).                   % 5
```


These two cases correspond to the situation where the sublist begins at the start of the list or the sublist begins later in the list, the only two possibilities.

Sample Executions

sublist([b,c,d], [a,b,c,d,e,f]) % 5

because sublist([b,c,d], [b,c,d,e,f]) % 2

because initialsublist([c,d], [c,d,e,f]) % 4

because initialsublist ([d], [d,e,f]) % 4

because initialsublist ([ ], [e,f]) % 3

sublist([b,d], [b,c,d]) fails % 2

because initialsublist([d], [c,d]) fails

and % 5

because sublist([b,d], [c,d]) fails % 5

because sublist([b,d], [d]) fails

Testing Primes in Prolog

Predicate prime

prime(P) succeeds iff P>0 is prime.

Assume the predicate
sqrt(N,S) iff S = floor(sqrt(N)).

```
prime(2).
prime(N) :- sqrt(N,S), okay(N,S,3).
```

where okay(N,S,D) succeeds iff no odd integer, D ≤ M ≤ S, divides into N evenly.

%----------

Predicate okay

```
okay(N,S,D) :- D>S.
okay(N,S,D) :-  N =\= D*(N//D), D1 is D+2,
                okay(N,S,D1).
```

%----------

Predicate sqrt

Consider this Wren program

```
program sqrt is
   var n,sqrt,odd,sum : integer;
begin
   read n;
   sqrt:=0; odd:=1; sum:=1;
   while sum<=n do
      sqrt:=sqrt+1;
      odd:=odd+2;
      sum:=sum+odd
   end while;
   write sqrt
end
```

Trace

n	sqrt	odd	sum
28	0	1	1
	1	3	4
	2	5	9
	3	7	16
	4	9	25
	5	11	36

Translate the while loop into Prolog as follows:

```
sqrt(N,S) :- loop(N,0,1,1,Ans).

loop(N,Sqrt,Odd,Sum,Ans) :-
                Sum =< N,
                Sqrt1 is Sqrt+1,
                Odd1 is Odd+2,
                Sum1 is Sum+Odd,
                loop(N,Sqrt1,Odd1,Sum1,Ans).
loop(N,Sqrt,Odd,Sum,Sqrt) :- Sum > N.
```

This last clause returns the value in the second parameter as the answer by unifying the last parameter with that second parameter.

%----------

List Processing

member(X,L) means item X is in list L

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

prefix(P,L) means list P is a prefix of list L

```
prefix([ ],L).
prefix([H|T],[H|U]) :- prefix(T,U).
```

suffix(S,L) means list S is a suffix of list L

```
suffix(S,S).
suffix(S,[H|T]) :- suffix(S,T).
```

sublist(Sb,L) means list Sb is a sublist of list L

```
sublist(Sb,L) :- prefix(Sb,L).
sublist(Sb,[H|T]) :- sublist(Sb,T).
```

concat(A,B,C) means list C is the concatenation of lists A and B

```
concat([ ], L, L).
concat([H|T], L, [H|M]) :- concat(T, L, M).
```

Alternative Definitions

```
prefix(P,L) :- concat(P,Q,L).

suffix(S,L) :- concat(R,S,L).

sublist(Sb,L) :- concat(A,Sb,C), concat(C,D,L).
```


take(N,L,NewL) means list NewL consists of the first N elements of list L

```
take(0,L,[ ]).
take(N,[H|T],[H|R]) :-  N>0, M is N-1,
                        take(M,T,R).
```

drop(N,L,NewL) means list NewL is L with its first N elements removed

```
drop(0,L,L).
drop(N,[H|T],R) :-  N>0, M is N-1,
                    drop(M,T,R).
```

nth(N,L,X) means item X is the Nth item in list L

```
nth(1,[H|T],H).
nth(N,[H|T],X) :- N>0, M is N-1, nth(M,T,X).
```

Factorial:

```
fac(0,1).
fac(N,F) :- N>0, N1 is N-1, fac(N1,R), F is N*R.
```


substitute(X,NX,L,NL) means NL is the list resulting from replacing all occurrences of item X in L by item NX

```
substitute(X,NX,[ ],[ ]).
substitute(X,NX,[X|T],[NX|U])
                    :- substitute(X,NX,T,U).
substitute(X,NX,[H|T],[H|U])
                    :- substitute(X,NX,T,U).
```

spliteven(L,Pre,Suf) means list L is split into a prefix and a suffix of the same length (±1)

```
len([ ],0).
len([H|T],N) :- len(T,M), N is M+1.

eq(M,N) :- N-2<M, M<N+2.

spliteven(L,Left,Right) :-  concat(Left,Right,L),
                            len(Left,M), len(Right,N),
                            eq(M,N).
```

Alternative: Try using take and drop.

nodups(L,NDL) means NDL is the list containing the items in L with all duplicates removed

```
nodups([ ],[ ]).
nodups([H|T],L) :-  member(H,T),
                    nodups(T,L).
nodups([H|T],[H|U]) :- nodups(T,U).
```

flatten(L,FlatL) means FlatL is the list L with all atomic items on the top level, for example flatten([a,[b,c],[[d],a]], [a,b,c,d,a])

Three predefined predicates:

```
atom(A) succeeds if A is an atom.
number(N) succeeds if N is a number.
atomic(X) :- atom(X).
atomic(X) :- number(X).

flatten([ ],[ ]).
flatten([H|T],[H|U]) :-  atomic(H), flatten(T,U).
flatten([H|T],R) :-  flatten(H,A), flatten(T,B),
                     concat(A,B,R).
```

Set Operations

subset(SubS,S) means every item in list SubS is also in list S

```
subset([ ],S).
subset([H|T],S) :- member(H,S),
                   subset(T,S).
```

union(L1,L2,Union) means list Union is the set union of L1 and L2 (no duplicates assuming L1 and L2 have none)

```
union([ ],S,S).
union([H|T],S,R) :-  member(H,S),
                     union(T,S,R).
union([H|T],S,[H|U]) :- union(T,S,U).
```

diff(L1,L2,Setdiff) means list Setdiff is the set difference of L1 and L2

```
diff([ ],S,[ ]).
diff([H|T],S,R) :- member(H,S), diff(T,S,R).
diff([H|T],S,[H|U]) :- diff(T,S,U).
```


Utility Predicates

get0(N)

N is bound to the ascii code of the next character from the current input stream (normally the terminal keyboard).

When the current input stream reaches its end of file, a special value is bound to N and the stream is closed.

26, the code for control-Z or

-1, a special end of file value.

put(N)

The character whose ascii code is the value of N is printed on the current output stream (normally the terminal screen).

see(F)

The file whose name is the value of F, an atom, becomes the current input stream.

seen

Close the current input stream.

write(T)

The Prolog term given by T is displayed on the current output stream.

tab(N)

N spaces are printed on the output stream.

nl

Newline prints a linefeed character on the current output stream.

abort

Immediately terminate the attempt to satisfy original query and return control to top level.

name(A,L)

A is a literal atom or a number, and L is a list of the ascii codes of the characters comprising the name of A.

```
| ?- name(A,[116,104,101]).
A = the

| ?- name(1994,L).
L = [49, 57, 57, 52]
```