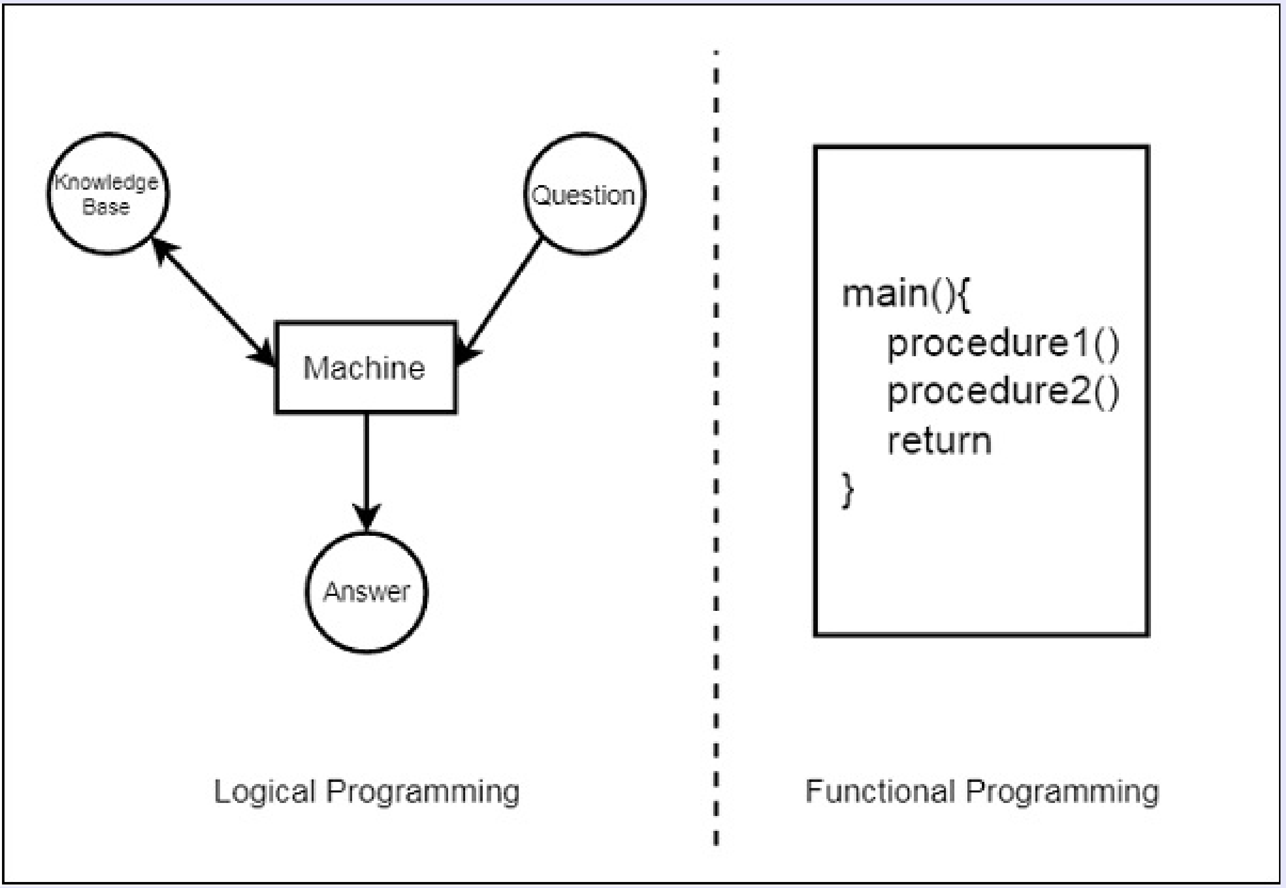
In the name of who surround me with blessing that I can not count

Prolog or PROgramming in LOGics

by `Ahmadreza` Anaami

Logic Programming is one of the Computer Programming Paradigm, in which the program statements express the facts and rules about different problems within a system of formal logic. Here, the rules are written in the form of logical clauses, where head and body are present. For example, H is head and B1, B2, B3 are the elements of the body. Now if we state that “H is true, when B1, B2, B3 all are true”, this is a rule. On the other hand, facts are like the rules, but without any body. So, an example of fact is “H is true”.

Some logic programming languages like Datalog or ASP (Answer Set Programming) are known as purely declarative languages. These languages allow statements about what the program should accomplish. There is no such step-by-step instruction on how to perform the task. However, other languages like Prolog, have declarative and also imperative properties. This may also include procedural statements like “To solve the problem H, perform B1, B2 and B3”.



```
main(){
   procedure1()
   procedure2()
   return
}
```

in the Logic Programming, we will provide **knowledge base**. Using this knowledge base, the machine can find answers to the given questions, which is totally different from functional programming.

In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the solution method.

Facts − The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

Rules − Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as −

grandfather(X, Y) :- father(X, Z), parent(Z, Y)

This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

Questions − And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

download GNU Prolog

http://www.gprolog.org/

lets begin

```
| ?- write(56).
56

yes
| ?- write('hello').
hello

yes
| ?- write('hello'),nl,write('world').
hello
world

yes
| ?- write("ABCDE")
.
[65,66,67,68,69]

yes
```

Conjunction

Conjunction (AND logic) can be implemented using the comma (,) operator.

Disjunction

Disjunction (OR logic) can be implemented using the semi-colon (;) operator.

```prolog
% Conjunction Logic
father(X,Y) :- parent(X,Y),male(X).
mother(X,Y) :- parent(X,Y),female(X).

% Disjunction Logic
child_of(X,Y) :- father(X,Y);mother(X,Y).
```

decision statements

```
% If-Then-Else statement

gt(X,Y) :- X >= Y,write('X is greater or equal').
gt(X,Y) :- X < Y,write('X is smaller').

% If-Elif-Else statement

gte(X,Y) :- X > Y,write('X is greater').
gte(X,Y) :- X =:= Y,write('X and Y are same').
gte(X,Y) :- X < Y,write('X is smaller').
```

loops

```
count_to_10(10) :- write(10),nl.
count_to_10(X) :-
   write(X),nl,
   Y is X + 1,
   count_to_10(Y).
```

list

- The first item, called the **head** of the list;
- The remaining part of the list, called the **tail**.

Suppose we have a list like: [red, green, blue, white, dark]. Here the head is red and tail is [green, blue, white, dark]. So the tail is another list.

Now, let us consider we have a list, L = [a, b, c]. If we write Tail = [b, c] then we can also write the list L as L = [a | Tail]. Here the vertical bar (|) separates the head and tail parts.

```
[a, b, c] = [x | [b, c] ]

[a, b, c] = [a, b | [c] ]

[a, b, c] = [a, b, c | [ ] ]
```

Membership Operation

```
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).
```

Length Calculation

```
list_length([],0).
list_length([_|TAIL],N) :- list_length(TAIL,N1), N is N1 + 1.
```

example

```
| ?- list_length([a,b,c,d,e,f,g,h,i,j],Len).

Len = 10

yes
| ?- list_length([],Len).

Len = 0
```

oncatenation

```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
```

- example

```
| ?- list_concat([1,2],[a,b,c],NewList).

NewList = [1,2,a,b,c]

yes
| ?- list_concat([],[a,b,c],NewList).

NewList = [a,b,c]

yes
| ?- list_concat([[1,2,3],[p,q,r]],[a,b,c],NewList).

NewList = [[1,2,3],[p,q,r],a,b,c]
```

Delete from List

```
list_delete(X, [X], []).
list_delete(X,[X|L1], L1).
list_delete(X, [Y|L2], [Y|L1]) :- list_delete(X,L2,L1).
```

```
| ?- list_delete(a,[a,e,i,o,u],NewList).

NewList = [e,i,o,u] ?

yes
| ?- list_delete(a,[a],NewList).

NewList = [] ?
```

Append into List

```
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).

list_append(A,T,T) :- list_member(A,T),!.
list_append(A,T,[A|T]).
```

In this case, we have used (!) symbol, that is known as cut. So when the first line is executed successfully, then we cut it, so it will not execute the next operation.

```
| ?- list_append(a,[e,i,o,u],NewList).

NewList = [a,e,i,o,u]

yes
| ?- list_append(e,[e,i,o,u],NewList).

NewList = [e,i,o,u]

yes
```

Order Operation

```
list_order([X, Y | Tail]) :- X =< Y, list_order([Y|Tail]).
list_order([X]).
```

```

yes
| ?- list_order([1,2,3,4,5,6,6,7,7,8]).

true ?


yes
| ?- list_order([1,4,2,3,6,5]).

no
| ?-
```

Divide List Operation

```
list_divide([],[],[]).
list_divide([X],[X],[]).
list_divide([X,Y|Tail], [X|List1],[Y|List2]) :-
   list_divide(Tail,List1,List2).
```

```
| ?- list_divide([a,1,b,2,c,3,d,5,e],L1,L2).

L1 = [a,b,c,d,e]
L2 = [1,2,3,5] ?

yes
| ?- list_divide([a,b,c,d],L1,L2).

L1 = [a,c]
L2 = [b,d]

yes
| ?-
```

max

```
max_of_two(X,Y,X) :- X >= Y.
max_of_two(X,Y,Y) :- X < Y.
```

```
list_max_elem([X],X).
list_max_elem([X,Y|Rest],Max) :-
   list_max_elem([Y|Rest],MaxRest),
   max_of_two(X,MaxRest,Max).
```

```
| ?- list_max_elem([8,5,3,4,7,9,6,1],Max).

Max = 9 ?

yes
```

list sum

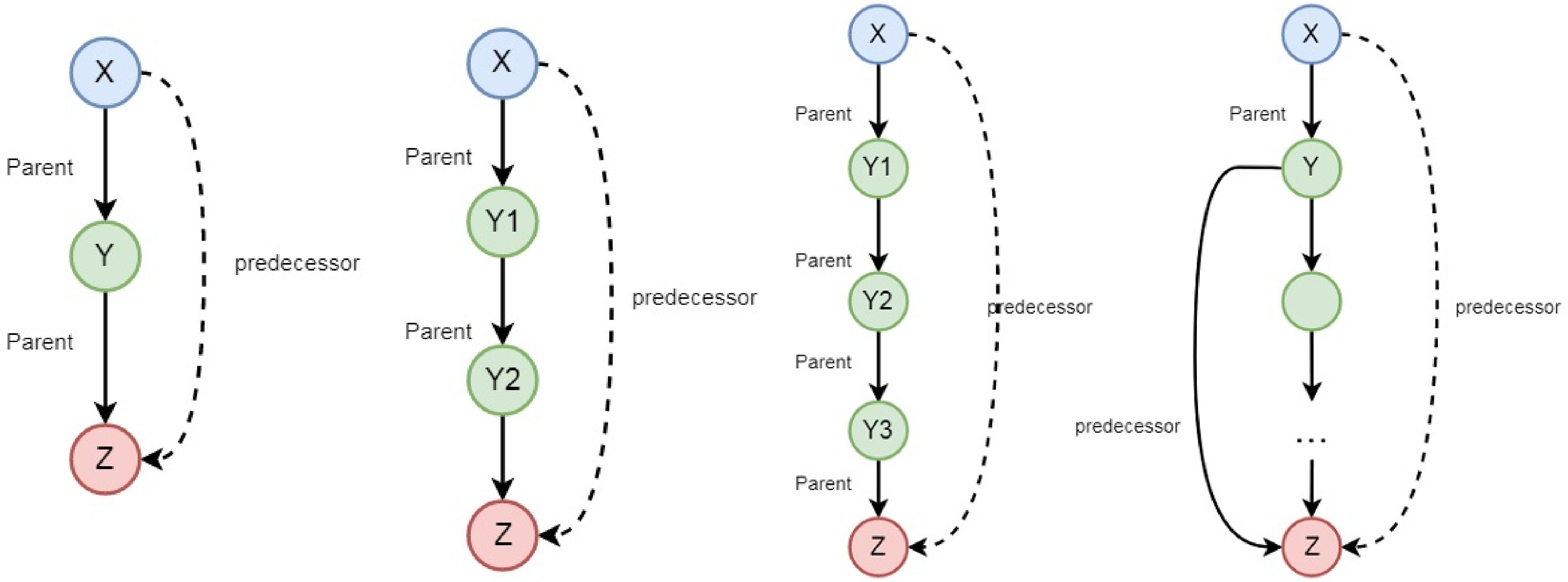
```
list_sum([],0).
list_sum([Head|Tail], Sum) :-
   list_sum(Tail,SumTemp),
   Sum is Head + SumTemp.
```

```
| ?- list_sum([5,12,69,112,48,4],Sum).

Sum = 250

yes
| ?- list_sum([8,5,3,4,7,9,6,1],Sum).

Sum = 43
```

extra info about Prolog lists

https://www.tutorialspoint.com/prolog/prolog_lists.htm

Recursion

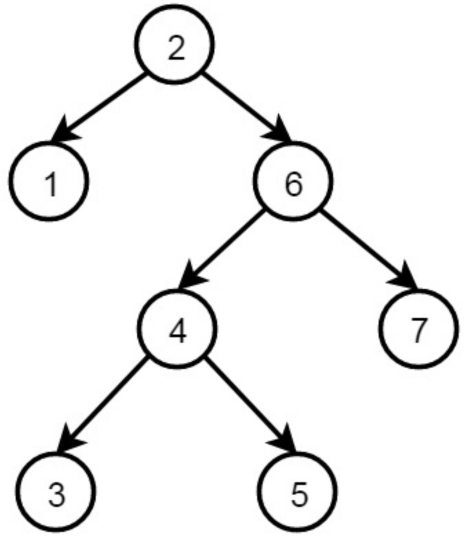

X
Parent
Y
Parent
Z
predecessor
X
Parent
Y1
Parent
Y2
Z
predecessor
X
Parent
Y1
Parent
Y2
Parent
Y3
Parent
Z
predecessor
X
Parent
Y
predecessor
...
Z
predecessor

Recursion

```prolog
predecessor(X, Z) :- parent(X, Z).

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

binary tree

```
node(2, node(1,nil,nil), node(6, node(4,node(3,nil,nil), node(5,nil,nil)), node(7,nil,nil))
```



Different and Not

different(X, Y) :- X = Y, !, fail ; true. % true is goal that always succeeds

```
different(X, X) :- !, fail.
different(X, Y).
```

```
| ?- different(100,100).

no
| ?- different(abc,def).

yes
```

```
different(X, Y) :- X = Y, !, fail ; true.
```

```
not(P) :- P, !, fail ; true.

| ?- not(true).

no
```

The read() Predicate

```
cube :-
   write('Write a number: '),
   read(Number),
   process(Number).
process(stop) :- !.
process(Number) :-
   C is Number * Number * Number,
   write('Cube of '),write(Number),write(': '),write(C),nl, cube.
```

```
| ?- cube.
Write a number: 2.
Cube of 2: 8
```

The tab() Predicate

```
Program
| ?- write('hello'),tab(15),write('world').
hello          world
yes

| ?- write('We'),tab(5),write('will'),tab(5),write('use'),tab(5),write('tabs').
We     will     use     tabs
yes

| ?-
```

Reading/Writing Files

```
| ?- tell('myFile.txt').

yes
| ?- tell('myFile.txt').

yes
| ?- write('Hello World').

yes
| ?- write(' Writing into a file'),tab(5),write('myFile.txt'),nl.

yes
| ?- write("Write some ASCII values").

yes
| ?- told. //we close the file
```

```
Hello World Writing into a file     myFile.txt
[87,114,105,116,101,32,115,111,109,101,32,65,83,67,73,73,32,118,97,108,117,101,115]
```

get more info

https://www.tutorialspoint.com/prolog/prolog_inputs_and_outputs.htm

Prolog - Built-In Predicates

https://www.tutorialspoint.com/prolog/prolog_built_in_predicates.htm

useful examples

https://www.tutorialspoint.com/prolog/prolog_basic_programs.htm