Prolog

Mohammed Alabbad
1082679
alabbama@mcmaster.ca
McMaster University
Department of Computing and Software

Content

- Introduction
- Prolog basics by example.
- Syntax.
- Data Structures.
- Backtracking and “cut”.
- Grammar Rules.
- References

Introduction

- Prolog is a general purpose logic programming language.
- It is a descriptive programming language (not conventional).
- Created by Alain Comerauer and Philippe Roussel in 1972.
- Originally aimed for natural languages processing.
- Now it is used in Artificial intelligence, expert systems, theorem proving.

Declarative Programming

- Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.
- Describing *what* the program should accomplish, rather than describing *how* to go about accomplishing it.
- Include languages of logic programming, functional programming, specific domain languages, and others.

Prolog as a logic programming language

- Logic programming: The use of mathematical logic for computer programming.

- The task is divided into two parts:
  - Program which contains objects and their relationships.
  - Queries which run against the program.

Prolog Example

- Declaring some facts (database) about objects and their relationships.

```
parent(john,mary).
female(mary).
```

- Asking questions about the facts.

```
?- parent(john,mary).
true.
```

```
?- parent(mary,john).
false.
```

- Variables can be instantiated or not instantiated.

```
parent(john,mary).
parent(karen,mary).
female(mary).
```

```
?- parent(john,X).
X = mary.
```

```
?- parent(X,mary).
X = john;
X = karen.
```

- Conjunctions are used for more complicated relationships.

```
parent(john,mary).
parent(karen,mary).
female(mary).
```

```
?- parent(john,mary),parent(karen,mary).
true.
```

```
?- parent(john,mary),parent(mary,john).
false.
```

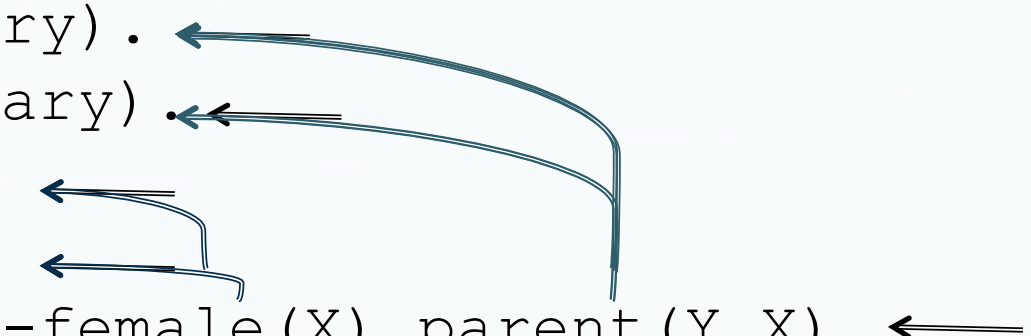
- Declaring some rules about objects and their relationships.

```
parent(john,mary). /*comment*/
parent(karen,mary).
female(mary).
daughter(X,Y):-female(X),parent(Y,X).
```

```
?- daughter(X,Y).
X = mary,
Y = john;
X = mary,
Y = karen.
```

- Backtracking.

```
parent(john,mary).
parent(karen,mary).
female(karen).
female(mary).
daughter(X,Y):-female(X),parent(Y,X).
```



```
?- daughter(X,Y).
X = mary,
Y = john;
X = mary,
Y = karen.
```

Basics

- Declaring facts.
- Declaring rules.
- Ask questions.
- Using conjunctions.
- An introduction to backtracking.

Syntax

- Data type:
  - Prolog has a single data type “term” which is either a constant, variable, or a structure.

1. Constants:
   - Constants name specific objects or specific relationships. there are two types: atoms and numbers.

- Atoms
  - a general-purpose name with no meaning.

```
parent john klm123 std_no ‘Gorge’ ‘1234’
:-  -?
```

- Numbers:
  - Integers or float.

```
0          20          1000          16.383
```

2. Variables

- indicated by a string consisting of letters and digits starting by Capital letter or underscore “_”.

```
X     Input      Annual_Income     _3789
```

```
parent(_,mary).
```

3. Structure “compound term”.
   - A structure is written by specifying its functor and components.

```
owns(john,book).
```

```
owns(john,book(prolog,gorge)).
```

```
owns(john,book(prolog,author(gorge,black))).
```

```
?- owns(john,book(X,author(_,_))).
X = prolog.
```

- Characters

```
A B C … Z
a b c … z
0 1 2 … 9
! " # $ % & ' ( ) = - ~ ^ | \ { } [ ] _ ` @
+ ; * : < > , . ? /
```

- Operators
  - Operators can be written as functors.

```
X+Y   +(X,Y)
```

  - Operators don’t cause any arithmetic carried out.

```
3+4 /*is not 7*/
```

```
plus(X,Y,X+Y).
```

```
?- plus(3,4,7).
false.
```

- Equality and Matching.

```
X=Y. /*equal*/                X/=Y./*not equal*/
```

- Arithmetic.
  - Arithmetic operations are used to compare numbers and calculate results.

```
=,==    \=,=\=    <    >    =<    >=
+       -         *    /    mod
is
```

- “is” infix operator to match the left-hand side with the right-hand side.

```
plus(X,Y,Z):- Z is X+Y.
```

```
?- plus(3,4,7).
true.
```

Data Structures

- Lists.
  - The list is a sequence of elements that can have any length. The elements of a list maybe any terms – constants, variables, structures-.
  - A list is either empty [] or has two elements: the head and the tail.

```
p([1,2,3]).
```

```
?- p([H|T]).
H = 1,
T = [2,3].
```

- Recursive Search.

```
member(X,[X|_]). /*boundary condition*/
member(X,[_|Y]):-member(X,Y).
```

```
?- member(a,[c,d,b,a]).
true.
```

```
?- member(a,[c,d,b,e]).
false.
```

- careful with recursive-looping.

```
parent(X,Y):-child(Y,X).
child(Y,X):-parent(X,Y).
```

- and left recursion.

```
person(X):-person(Y),mother(Y,X).
perosn(mary).
mother(mary,john).
```

```
?- person(X).
ERORR:Out of local stack.
```

- Joining Structures:

```
append([],L,L).
append([X|L1],L2,[X|L3]):-append(L1,L2,L3).
```

```
?- append([a,b,c],[e,f,g],X).
X = [a,b,c,e,f,g].
```

```
?- append(X,[e,f,g],[a,b,c,e,f,g]).
X = [a,b,c].
```

backtracking

- An attempt to satisfy the goal, from top to down.
  - A match found. Prolog marks it and instantiates any uninstantiated variables. If it is a rule Prolog tries to satisfy the subgoals.
  - No match found. Prolog fails and attempts to re-satisfy the left goal, unistanstating the variables instantiated by the goal.

- Generating multiple solutions.

```
parent(john,mary).
parent(karen,mary).
parent(john,mike).
parent(karen,mike).
```

```
parent(X,_).
X = john;
X = karen;
X = john;
X = karen.
```

Cut

- cut “!” tells Prolog which previous choices need not to be considered when it backtracks. It is said it is important for two reasons: faster and less memory space used.

```
foo:-a,b,c,!,d,e,f.
foo:-g,h.
```

Common uses of Cut

- Confirm the choice of rules.

```
sum_to(1,1):-!.
sum_to(N,Res):-N1 is N - 1,
               sum_to(N1,Res1),
               Res is Res1 + N.
```

```
sum_to(5,X).
X = 15.
```

```
sum_to(1,1).
sum_to(N,Res):-not(N =< 1),
               N1 is N - 1,sum_to(N1,Res1),
               Res is Res1 + N.
```

- “cut-fail” combination.

```
foo(f):-!,fail.
foo(X):-a,b,c.
```

Problems with Cut

- The way Prolog searches the database should be taking into account, because “cut” could have strange behaviour if used in another way.

```
append([],X,X):-!.
append([A|B],C,[A|D]:-append(B,C,D).
```

```
?- append(X,Y,[a,b,c]).
X = [],
Y = [a, b, c].
```

```
number_of_parents(adam,0):-!.
number_of_parents(eve,0) :-!.
number_of_parents(X,2).
```

```
?- number_of_parents(eve,X).
X = 0.
?- number_of_parents(john,X).
X = 2.
?- number_of_parents(eve,2).
true.
```

“cut” should be avoided

- knowing how backtracking satisfies all possibilities.
- So, if you introduce “cut” there is no guarantee that anything sensible will happen if another goals start appearing.
- “cut” should be avoided because how rules will be used is not clear.

Grammar Rules

```
<sentence> ::= <noune_phrase> <verb_phrase>
<noune_phrase> ::= <determiner> <noun>
<verb_phrase> ::= <verb> <noune_phrase>
<verb_phrase> ::= <verb>
<determinar> ::= [the]
<noun> ::= [apple]
<noun> ::= [man]
<verb> ::= [eats]
<verb> ::= [sings]
```

```
sentence --> sentence(X).
sentence(X) --> noun_phrase(X),verb_phrase(X).

noun_phase(X)--> determinar(X), noun(X).

verb_phrase(X) --> verb(X).
verb_phrase(X) --> verb(X), noun_phras(Y).

noun(singular) --> [boy].
noun(plural) --> [boys].
determiner(_) --> [the].
verb(singular) --> [eat].
verb(plural) --> [eats].
```

References

- Clocksin, W. F., and C. S. Mellish. *Programming in Prolog*. Berlin: Springer-Verlag, 1994.
- "Prolog." *Wikipedia, the Free Encyclopedia*. Web. 23 Oct. 2010. <http://en.wikipedia.org/wiki/Prolog>.
- "Logic Programming." *Wikipedia, the Free Encyclopedia*. Web. 23 Oct. 2010. < http://en.wikipedia.org/wiki/Logic_programming>.
- "Declarative Programming." *Wikipedia, the Free Encyclopedia*. Web. 23 Oct. 2010. < http://en.wikipedia.org/wiki/ Declarative_programming>.

Thank you!