Logic and Functional Programming

Basic Elements of Prolog

Dr. Cristian-Paul Bara

Computer Science Department
Faculty of Mathematics and Computer Science,
Babeş Bolyai University,
Cluj-Napoca, Romania

Table of Contents

Prolog Trivia

- Programmation LOGique, developed in Marseille in 19070
- Language standard: ISO-Prolog
- Objecct oriented versions exist, though not standardized
- SWI-Prolog, a close implementation of the standard, will be used in this course
- SWI-Prolog 1986 offers bidirectional interfacing with C and Java

Parts of the Prolog Program

Facts or Clauses

```
father(joe,paul).
father(joe,mary).
father(joe,hope).
mother(jane,paul).
mother(jane,mary).
mother(jane,hope).
male(paul).
female(mary).
female(hope).
```

Parts of the Prolog Program

Rules or Predicates

```
male(X) :- father(X,_).
female(X) :- mother(X,_).
sibling(X,Y) :- father(Z,X),father(Z,Y),X\ =Y.
sibling(X,Y) :- mother(Z,X),mother(Z,Y),X\ =Y.
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
child(X,Y) :- parent(Y,X).
son(X,Y) :- parent(Y,X),male(X).
daughter(X,Y) :- parent(Y,X),female(X).
```

Parts of the Prolog Program

Goals

```
pauls_father :- father(Pauls_father,paul), write("Paul’s
father is "), write(Pauls_father).
janes_son :- son(Janes_son,jane), write("Jane’s son is "),
write(Janes_son).
```

Prolog and First Order Logic

First order Logic	Prolog
$\forall x, p(x) \wedge q(x) \rightarrow r(x)$	r(X):-p(X),q(X).
$\forall x, w(x) \vee s(x) \rightarrow p(x)$	p(X):-w(X).
	p(X):-s(X).
$\forall x, t(x) \rightarrow s(x) \wedge q(x)$	s(X):-t(X).
	q(X):-t(X).

Food for thought

What is the prolog equivalent for:

$$\forall x, s(x) \rightarrow p(x) \vee q(x)$$

Equality operators

X=Y checks if X and Y can be unified

```
?-[a,b]=[a,b]   ?-[X,Y]=[a,b]   ?-[a,b]=[X,Y]
true            X=a             X=a
                Y=b             Y=b
```

X\=Y, \+ X=Y checks if X and Y cannot be unified

```
?-[X,Y,Z]\=[b,a]   ?-[X,Y]\=[a,b]      ?-[a,b]\=[X,Y]
true               false               false

?-\+a=b            ?-\+[X,Y]=[a,b]     ?-\+[a,b]=[X,Y]
false              false               false
```

Equality operators

X==Y checks if X and Y are bounded to the same value

?-[2,3]==[2,3]	?-a==a	R==1
true	**true**	**false**

X==Y checks if X and Y are bounded to the same value

?-[2,3]\==[3,2]	?-a\==a	R\==1
true	**false**	**true**

Arithmetic operators

=:= checks for arithmetic equality, evaluates both sides, both sides must be numeric, variables are bounded
=\= checks for arithmetic inequality
is the right side is bounded to the left side, the left side must be a variable, if variable is bounded arithmetic equality is checked, if not the right side is evaluated and ”assigned”to the left side

Arithmetic operators

mod modulo division, returns remainder
div returnd floor of division
abs(X) returns absolute value of X
sqrt(X) returns square root of X
round(X) returns the nearest integer of X

Preddefined predicates

var(X) true if X is free, false if bounded
number(X) true if X is bounded to a number
integer(X) true if X is bounded to an integer
float(X) true if X is bounded to a real number
atom(X) true if X is bounded to an atom
atomic(X) number(X) or atom(X)
is_list(X) checks if X is a list

Predefined Domains

char: character between single quotes, e.g., ’c’

integer: a whole number between -32768 and 32767

real: a floating point number between 10^{-307} and 10^{308}

string: a character sequence between double quotes, e.g., ”string of characters”

symbol:

1. sequence of letters, numbers and underscores starting with a lower case letter, e.g., symbol_name
2. sequence of characters between double quotes, e.g., ”name of a symbol”

Multiple arity predicates

```
mother(jane,hope).
mother(X) :- mother(X,_).
```

The IF symbol (Prolog) and the IF instruction (other languages)

- Implicit then/if:

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

- Explicit if/then/else:

  ```
  sign(X) :- X>=0 -> write(X), write(’ is positive’);
  write(X), write(’ is negative’).
  ```

Compiler directives

include: include other source files

trace: the system will output all predicate calls in the execution of the program

shorttrace: the system will output a subset predicate calls in the execution of the program based on compiler optimization

Arithmetic expressions and comparisons

```
solve(A,B,C) :- D is B*B-4*A*C,respond(A,B,D),nl.
respond(_,_,D) :- D<0,write("There are no solutions").
respond(A,B,D) :- D=:=0, X is -B/(2*A),write("x = "),
write(X).
respond(A,B,D) :-
  D>0,
  SqrtD is sqrt(D),
  X1 is (-B-SqrtD)/(2*A),
  X2 is (-B+SqrtD)/(2*A),
  write("x1 = "), write(X1), write(" x2 = "), write(X2).
goal :- solve(1,2,0).
```

Input/output operations. Strings

```
solve(A,B,C) :- D is B*B-4*A*C,respond(A,B,D),nl.
respond(_,_,D) :- D<0,write("There are no solutions").
respond(A,B,D) :- D=:=0, X is -B/(2*A),write("x = "),
write(X).
respond(A,B,D) :-
  D>0,
  SqrtD is sqrt(D),
  X1 is (-B-SqrtD)/(2*A),
  X2 is (-B+SqrtD)/(2*A),
  write("x1 = "), write(X1), write(" x2 = "), write(X2).
goal :- read(X), read(Y), read(Z), solve(X,Y,Z).
```

The findall predicate

```
findall(Arg1,Arg2,Arg3)
```

Arg1 specifies the values to be collected

Arg2 specifies the predicate

Arg3 specifies the list to be created

```
p(a,b)
p(b,c)
p(a,c)
p(a,d)
all(X,L):-findall(Y,p(X,Y),L)
?-all(X,L)
L=[b,c,d]
```

Lists

- Lists are tuples of ”head of list” and ”tail of list”
- equivalent reprezentations: [a,b,c] [a|[b,c]] [a|[b|[c]]] [a|[b|[c|[]]]]
- binding [H|T] to [a,b,c] result in H=a and T=[b,c]
- binding [H1|[H2|T]] to [a,b,c] results in H1=a, H2=b, and T=[c]