# Some Built-in Predicates

## 1. Arithmetic Operators, Equality, and Comparisons

X is E True if X matches the value of the arithmetic expression E;  
 Normally, X is a variable and the result of matching will have   
 X bound to the result of evaluating E. Therefore, E must be   
 an arithmetic expression.   
  
 X = Y True if X and Y are unifiable  
  
 E1 =:= E2 True if the values of arithmetic exps E1 and E2 are equal  
  
 E1 =\= E2 True if the values of arithmetic exps E1 and E2 are not equal  
  
 T1 == T2 True if T1 and T2 are identical  
  
 T1 \== T2 True if T1 and T2 are not identical   
  
 Except the case of identical var on both sides, e.g. A \== A,   
 the statement is true if either variable T1 or T2 are unbound,   
 or if they are both bound, but to different structures.   
 The expression fails if both variables are bound to  
 the same constant (see pages 175-176 of Bratko).

Examples:

| ?- N is 3+6.  
N = 9

| ?- N is X+5.      % expression must be evaluable, but in this example X is unbound.  
! Instantiation error in argument 2 of is/2  
! goal:  \_76 is \_73+5  
  
| ?- N is 3, M is N +5.  
M = 8,  
N = 3   
  
| ?- N = f(X).  
N = f(X)

| ?- 4+5 =:= 9.  
yes

| ?- X == X.  
true

| ?-  X == x.  
no

| ?-  f(X) \== N.  
true   
  
| ?- 2+4 \== 7.  
yes  
  
Typical arithmetic and comparison operators

+ - \* /  
 <, >, =<, >=

## 2. Meta-logical predicates

var(X): tests whether X is uninstantiated.  
 nonvar(X): opposite of var.  
 atom(X): checks if X is instantiated to an atom.  
 integer(X):  
 number(X):  
 atomic(X): true if X is either an atom or a number

Examples.

var(f(Z)) -> fails, since the argument is a structure f(Z)  
 var(Z) -> succeeds  
 nonvar(Z) -> fails  
 nonvar(f(a,N)) -> succeeds  
 atom(23) -> fails  
 atom(iden) -> succeeds  
 number(1.0) -> succeeds  
 integer(1.0) -> fails  
  
  
Negation \+: E.g., if you have defined member/2, you   
can write  
  
 \+ member(...)  
  
which holds if the attempt to prove member(...) fails.  
  
More on negation later.  
  
  
You may choose to directly define a negated predicate; e.g.,  
for "not member", you can define:  
  
notMember(\_,[]).  
notMember(A,[B|R]) :-   
 A \== B,  
 notMember(A,R).

## 3. Finding all solutions

Prolog usually generates solutions one by one with backtracking. Sometimes it is useful to collect all solutions in a list. There are three built-in Prolog predicates that can be used for this purpose. They are called bagof, setof and findall.

Let's briefly explain how findall works. In the Bratko book it is on page 168-9. Click [here](http://www.sics.se/sicstus/docs/3.7.1/html/sicstus_10.html" \l "SEC111) from the manual page of sicstus Prolog.

findall(X,P,L)

returns a list L with all values for X that satisfy predicate P. P should contain the same variable X somewhere.

Examples:

Let the database be  
likes(bill,cider).  
likes(dick,beer).  
likes(harry,beer).  
likes(jan,cider).  
likes(tom,cider).  
likes(tom,beer).

Then:

?- findall(X,likes(X,beer),L).  
L = [dick,harry,tom]  
  
?- findall(X,likes(X,Y),L).  
L = [bill,dick,harry,tom,tom,jan]

Note that the goal

likes(X,Y)

in the query means "there exists a Y such that likes(X,Y)". Thus, "free variables" in a goal of the findall predicate are existentially quantified.

In general, X in findall(X,P,L) can be a *template*.

E.g.

?- findall((X,Y),likes(X,Y),L).  
  
L = [(bill,cider),(dick,beer),(harry,beer),(tom,beer),(tom,cider),(jan,cider)]   
  
?- findall(a(X),likes(X,beer),L).  
  
L = [a(dick),a(harry),a(tom)]

Last modified: Tuesday, 30 October 2012, 11:24 AM