Prolog - Built-In Predicates

In Prolog, we have seen the user defined predicates in most of the cases, but there are some built-in-predicates. There are three types of built-in predicates as given below –

- Identifying terms
- Decomposing structures
- Collecting all solutions

So this is the list of some predicates that are falls under the identifying terms group –

Predicate	Description
var(X)	succeeds if X is currently an un-instantiated variable.
novar(X)	succeeds if X is not a variable, or already instantiated
atom(X)	is true if X currently stands for an atom
number(X)	is true if X currently stands for a number
integer(X)	is true if X currently stands for an integer
float(X)	is true if X currently stands for a real number.
atomic(X)	is true if X currently stands for a number or an atom.
compound(X)	is true if X currently stands for a structure.
ground(X)	succeeds if X does not contain any un-instantiated variables.

The var(X) Predicate

When X is not initialized, then, it will show true, otherwise false. So let us see an example.

Example

```
| ?- var(X).

yes
```

```
| ?- X = 5, var(X).

no
| ?- var([X]).

no
| ?-
```

The novar(X) Predicate

When X is not initialized, the, it will show false, otherwise true. So let us see an example.

Example

```
| ?- nonvar(X).

no
| ?- X = 5,nonvar(X).

X = 5

yes
| ?- nonvar([X]).

yes
| ?-
```

The atom(X) Predicate

This will return true, when a non-variable term with 0 argument and a not numeric term is passed as X, otherwise false.

Example

```
| ?- atom(paul).
```

```
yes
| ?- X = paul,atom(X).

X = paul

yes
| ?- atom([]).

yes
| ?- atom([a,b]).

no
| ?-
```

The number(X) Predicate

This will return true, X stands for any number, otherwise false.

Example

```
| ?- number(X).

no
| ?- X=5,number(X).

X = 5

yes
| ?- number(5.46).

yes
| ?-
```

The integer(X) Predicate

This will return true, when X is a positive or

negative integer value, otherwise false.

Example

```
| ?- integer(5).

yes
| ?- integer(5.46).

no
| ?-
```

The float(X) Predicate

This will return true, X is a floating point number, otherwise false.

Example

```
| ?- float(5).

no
| ?- float(5.46).

yes
| ?-
```

The atomic(X) Predicate

We have atom(X), that is too specific, it returns false for numeric data, the atomic(X) is like atom(X) but it accepts number.

Example

```
| ?- atom(5).

no
```

```
| ?- atomic(5).

yes
| ?-
```

The compound(X) Predicate

If atomic(X) fails, then the terms are either one non-instantiated variable (that can be tested with var(X)) or a compound term. Compound will be true when we pass some compound structure.

Example

```
| ?- compound([]).

no
| ?- compound([a]).

yes
| ?- compound(b(a)).

yes
| ?-
```

The ground(X) Predicate

This will return true, if X does not contain any un-instantiated variables. This also checks inside the compound terms, otherwise returns false.

Example

```
| ?- ground(X).
```

```
no
| ?- ground(a(b,X)).

no
| ?- ground(a).

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

yes
| ?-
```

Decomposing Structures

Now we will see, another group of built-in predicates, that is Decomposing structures. We have seen the identifying terms before. So when we are using compound structures we cannot use a variable to check or make a functor. It will return error. So functor name cannot be represented by a variable.

Error

```
X = tree, Y = X(maple).
Syntax error Y=X<>(maple)
```

Now, let us see some inbuilt predicates that falls under the Decomposing structures group.

The functor(T,F,N) Predicate

This returns true if F is the principal functor of T, and N is the arity of F.

Note – Arity means the number of attributes.

Example

```
| ?- functor(t(f(X),a,T),Func,N).

Func = t
N = 3

(15 ms) yes
| ?-
```

The arg(N,Term,A) Predicate

This returns true if A is the Nth argument in Term. Otherwise returns false.

Example

```
| ?- arg(1,t(t(X),[]),A).

A = t(X)

yes
| ?- arg(2,t(t(X),[]),A).

A = []

yes
| ?-
```

Now, let us see another example. In this example, we are checking that the first argument of D will be 12, the second argument will be apr and the third argument will be 2020.

Example

```
| ?- functor(D,date,3), arg(1,D,12

D = date(12,apr,2020)

yes
| ?-
```

The ../2 Predicate

This is another predicate represented as double dot (..). This takes 2 arguments, so '/2' is written. So Term = .. L, this is true if L is a list that contains the functor of Term, followed by its arguments.

Example

```
| ?- f(a,b) =.. L.

L = [f,a,b]

yes
| ?- T =.. [is_blue,sam,today].

T = is_blue(sam,today)

yes
| ?-
```

By representing the component of a structure as a list, they can be recursively processed without knowing the functor name. Let us see another example –

Example

```
| ?- f(2,3)=..[F,N|Y], N1 is N*3,
```

```
F = f
L = f(6,3)
N = 2
N1 = 6
Y = [3]

yes
| ?-
```

Collecting All Solutions

Now let us see the third category called the collecting all solutions, that falls under built-in predicates in Prolog.

We have seen that to generate all of the given solutions of a given goal using the semicolon in the prompt. So here is an example of it.

Example

```
| ?- member(X, [1,2,3,4]).

X = 1 ? ;

X = 2 ? ;

X = 3 ? ;

X = 4

yes
```

Sometimes, we need to generate all of the solutions to some goal within a program in some AI related applications. So there are three built-in predicates that will help us to

get the results. These predicates are as follows –

- findall/3
- setoff/3
- bagof/3

These three predicates take three arguments, so we have written '/3' after the name of the predicates.

These are also known as **meta-predicates**. These are used to manipulate Prolog's Proof strategy.

Syntax

```
findall(X,P,L).
setof(X,P,L)
bagof(X,P,L)
```

These three predicates a list of all objects X, such that the goal P is satisfied (example: age(X,Age)). They all repeatedly call the goal P, by instantiating variable X within P and adding it to the list L. This stops when there is no more solution.

Findall/3, Setof/3 and Bagof/3

Here we will see the three different built-in predicates findall/3, setof/3 and the bagof/3, that fall into the category, **collecting all solutions**.

The findall/3 Predicate

This predicate is used to make a list of all

solutions X, from the predicate P. The returned list will be L. So we will read this as “find all of the Xs, such that X is a solution of predicate P and put the list of results in L”. Here this predicate stores the results in the same order, in which Prolog finds them. And if there are duplicate solutions, then all will come into the resultant list, and if there is infinite solution, then the process will never terminate.

Now we can also do some advancement on them. The second argument, which is the goal, that might be a compound goal. Then the syntax will be as **findall(X, (Predicate on X, other goal), L)**

And also the first argument can be a term of any complexity. So let us see the examples of these few rules, and check the output.

Example

```
| ?- findall(X, member(X, [1,2,3,4

Results = [1,2,3,4]

yes
| ?- findall(X, (member(X, [1,2,3,

Results = [3,4]

yes
| ?- findall(X/Y, (member(X,[1,2,3

Results = [1/1,2/4,3/9,4/16]

yes
| ?-
```

The setof/3 Predicate

The setof/3 is also like findall/3, but here it removes all of the duplicate outputs, and the answers will be sorted.

If any variable is used in the goal, then that will not appear in the first argument, setof/3 will return a separate result for each possible instantiation of that variable.

Let us see one example to understand this setof/3. Suppose we have a knowledge base as shown below –

```
age(peter, 7).
age(ann, 5).
age(pat, 8).
age(tom, 5).
age(ann, 5).
```

Here we can see that age(ann, 5) has two entries in the knowledge base. And the ages are not sorted, and names are not sorted lexicographically in this case. Now let us see one example of setof/3 usage.

Example

```
| ?- setof(Child, age(Child,Age),R

Age = 5
Results = [ann,tom] ? ;

Age = 7
Results = [peter] ? ;

Age = 8
Results = [pat]
```

```
(16 ms) yes
| ?-
```

Here we can see the ages and the names both are coming sorted. For age 5, there is two entries, so the predicate has created one list corresponding to the age value, with two elements. And the duplicate entry is present only once.

We can use the nested call of setof/3, to collect together the individual results. We will see another example, where the first argument will be Age/Children. As the second argument, it will take another setof like before. So this will return a list of (age/Children) pair. Let us see this in the prolog execution –

Example

```
| ?- setof(Age/Children, setof(Chi

AllResults = [5/[ann,tom],7/[peter

yes
| ?-
```

Now if we do not care about a variable that does not appear in the first argument, then we can use the following example –

Example

```
| ?- setof(Child, Age^age(Child,Ag

Results = [ann,pat,peter,tom]

yes
```

```
| ?-
```

Here we are using the upper caret symbol (^), this indicates that the Age is not in the first argument. So we will read this as, “Find the set of all children, such that the child has an Age (whatever it may be), and put the result in Results”.

The bagof/3 Predicate

The bagof/3 is like setof/3, but here it does not remove the duplicate outputs, and the answers may not be sorted.

Let us see one example to understand this bagof/3. Suppose we have a knowledge base as follows –

Knowledge Base

```
age(peter, 7).
age(ann, 5).
age(pat, 8).
age(tom, 5).
age(ann, 5).
```

Example

```
| ?- bagof(Child, age(Child,Age),R

Age = 5
Results = [ann,tom,ann] ? ;

Age = 7
Results = [peter] ? ;

Age = 8
Results = [pat]
```

```
(15 ms) yes
| ?-
```

Here for the Age value 5, the results are [ann, tom, ann]. So the answers are not sorted, and duplicate entries are not removed, so we have got two 'ann' values.

The bagof/3 is different from findall/3, as this generates separate results for all the variables in the goal that do not appear in the first argument. We will see this using an example below –

Example

```
| ?- findall(Child, age(Child,Age)

Results = [peter,ann,pat,tom,ann]

yes
| ?-
```

Mathematical Predicates

Following are the mathematical predicates –

Predicates	Description
random(L,H,X).	Get random value between L and H
between(L,H,X).	Get all values between L and H
succ(X,Y).	Add 1 and assign it to X
abs(X).	Get absolute value of X
max(X,Y).	Get largest value between X and Y
min(X,Y).	Get smallest value between X and Y
round(X).	Round a value near to X
truncate(X).	Convert float to integer, delete the fractional part
loor(X).	Round down
ceiling(X).	Round up
sqrt(X).	Square root

Besides these, there are some other predicates such as sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh, log, log10, exp, pi, etc.

Now let us see these functions in action using a Prolog program.

Example

```
| ?- random(0,10,X).

X = 0

yes
```

```
| ?- random(0,10,X).

X = 5

yes
| ?- random(0,10,X).

X = 1

yes
| ?- between(0,10,X).

X = 0 ? a

X = 1

X = 2

X = 3

X = 4

X = 5

X = 6

X = 7

X = 8

X = 9

X = 10

(31 ms) yes
| ?- succ(2,X).

X = 3

yes
| ?- X is abs(-8).
```

```
X = 8

yes
| ?- X is max(10,5).

X = 10

yes
| ?- X is min(10,5).

X = 5

yes
| ?- X is round(10.56).

X = 11

yes
| ?- X is truncate(10.56).

X = 10

yes
| ?- X is floor(10.56).

X = 10

yes
| ?- X is ceiling(10.56).

X = 11

yes
| ?- X is sqrt(144).

X = 12.0

yes
| ?-
```

Print Page