CS370

Symbolic Programming
Declarative Programming

LECTURE 9: More Built-in Predicates

Jong C. Park

park@cs.kaist.ac.kr

Computer Science Department

Korea Advanced Institute of Science and Technology

http://nlp.kaist.ac.kr/~cs370

More Built-in Predicates

- Testing the type of terms
- Constructing and decomposing terms
- Equality and comparison
- Database manipulation
- Control facilities
- *bagof*, *setof* and *findall*

Testing the type of terms

- *var, nonvar, atom, integer, float, number, atomic, compound*

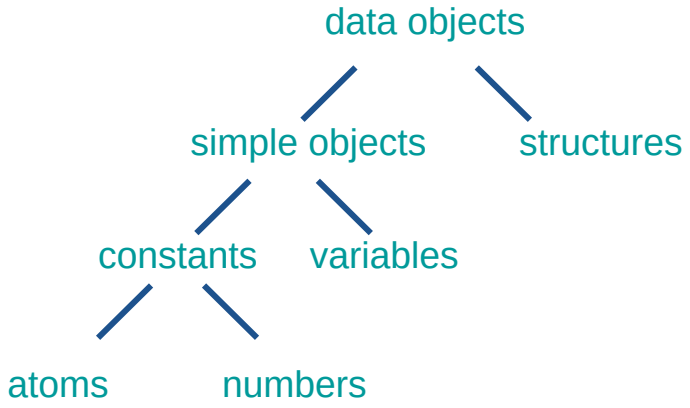
```
Z is X + Y
..., number(X), number(Y), Z is X+Y, ...
?- var(Z), Z = 2.
Z = 2
?- Z = 2, var(Z).
no
?- integer(Z), Z = 2.
no
?- Z = 2, integer(Z), nonvar(Z).
Z = 2
```

Testing the type of terms

- *var, nonvar, atom, integer, float, number, atomic, compound*

```
?- atom(3.14).
no
?- atomic(3.14).
yes
?- atom(==>).
yes
?- atom(p(1)).
no
?- compound(2+X).
true
```

number or atom



Testing the type of terms

- Example: count(A,L,N)
  - Sample Program

```
count(_,[ ],0).
count(A,[A|L],N) :- !, count(A,L,N1), N is N1 + 1.
count(A,[_|L],N) :- count(A,L,N).
```

  - Problem?

```
?- count(a,[a,b,a,a],N).
N = 3
?- count(a,[a,b,X,Y],Na).
Na = 3
```

Testing the type of terms

- Example: count(A,L,N)

```
?- count(b,[a,b,X,Y],Nb).
Nb = 3
?- L = [a,b,X,Y], count(a,L,Na), count(b,L,Nb).
Na = 3
Nb = 1
X = a
Y = a
```

Testing the type of terms

- Example: count(A,L,N)
  - Revised Program

```
count(_,[ ],0).
count(A,[B|L],N) :- atom(B), A = B, !,
                    count(A,L,N1), N is N1+1
                    ;
                    count(A,L,N).
```

  - Problem?

```
?- count(2,[a,a,2,b],N).
N = 0
?- count(A,[A,B,A,A],N).
```

Testing the type of terms

- A cryptarithmetic puzzle using *nonvar*

```
   D O N A L D
+  G E R A L D
-------------
   R O B E R T
```

sum(N1,N2,N)

constraints: all letters are assigned different digits

```
sum([D,O,N,A,L,D],
    [G,E,R,A,L,D],
    [R,O,B,E,R,T])
```

Testing the type of terms

- A cryptarithmetic puzzle using *nonvar*
  - necessary information
    - carry digits before and after the summation
    - set of available digits before the summation
    - remaining digits
    - N1, N2 and N

sum1(N1, N2, N, C1, C, Digits1, Digits)

Testing the type of terms

- A cryptarithmetic puzzle using *nonvar*

```
?-
   sum1([H,E],[6,E],[U,S],1,1,[1,3,4,7,8,9],Digits).
H = 8
E = 3
S = 7
U = 4
Digits = [1,9]
```

Testing the type of terms

- A cryptarithmetic puzzle using *nonvar*

```
sum(N1,N2,N) :-
   sum1(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],_).

sum1([D1|N1],[D2|N2],[D|N],C1,C,Digs1,Digs) :-
   sum1(N1,N2,N,C1,C2,Digs1,Digs2),
   digitsum(D1,D2,C2,D,C,Digs2,Digs).

del_var(Item,List,List) :- nonvar(Item), !.
del_var(Item,[Item|List],List).
del_var(Item,[A|List],[A|List1] :-
   del_var(Item,List,List1).

?- puzzle1(N1,N2,N), sum(N1,N2,N).
```

Constructing and decomposing terms

- =.., *functor, arg, name*

  Term =.. L

  ?- f(a,b) =.. L.
  L = [f, a, b]
  ?- T =.. [rectangle,3,5].
  T = rectangle(3,5)
  ?- Z =.. [p,X,f(X,Y)].
  Z = p(X,f(X,Y))

Constructing and decomposing terms

- =.., *functor, arg, name*

```
square(Side)
triangle(Side1,Side2,Side3)
circle(R)

enlarge(Fig,Factor,Fig1)

enlarge(square(A),F,square(A1)) :-
    A1 is F*A.
enlarge(Type(Par),F,Type(Par1)) :-
    Par1 is F*Par.
```

Constructing and decomposing terms

- =.., *functor, arg, name*
  - Sample Program

```
enlarge(Fig,F,Fig1) :-
    Fig =.. [Type|Pars],
    multiplylist(Pars,F,Pars1),
    Fig1 =.. [Type|Pars1].
multiplylist([ ],_,[ ]).
multiplylist([X|L],F,[X1|L1]) :-
    X1 is F*X, multiplylist(L,F,L1).
```

Constructing and decomposing terms

- =.., *functor, arg, name*

```
% substitute(Subterm,Term,Subterm1,Term1)

?- substitute(sin(x), 2*sin(x)*f(sin(x)), t, F).
F = 2*t*f(t)
?- substitute(a+b, f(a,A+B), v, F).
F = f(a,v)
A = a
B = b
```

Figure 7.3

Constructing and decomposing terms

- =.., *functor, arg, name*

```
obtain(Functor),
compute(Arglist),

Goal =.. [Functor|Arglist],
Goal.
...
Goal =.. [Functor|Arglist],
call(Goal).
```

Constructing and decomposing terms

- *=.., functor, arg, name*

```
?- functor(t(f(X),X,t), Fun, Arity).
Fun = t
Arity = 3
?- arg(2, f(X, t(a), t(b)), Y).
Y = t(a)
?- functor(D,date,3), arg(1,D,29), arg(2,D,june),
   arg(3,D,1982).
D = date(29, june, 1982)
```

Equality and comparison

- Equality

```
X = Y
X is E
E1 =:= E2 (E1 =\= E2)
T1 == T2 (T1 \== T2)
?- f(a,b) == f(a,b).
yes
?- f(a,b) == f(a,X).
no
?- f(a,X) = f(a,Y).
X = Y
?- X \== Y.
?- t(X,f(a,Y)) == t(X,f(a,Y)).
```

Equality and comparison

- Equality
  - Revised Program

```
count(_,[ ],0).
count(Term, [Head|L], N) :-
   Term == Head, !,
   count(Term, L, N1),
   N is N1+1
   ;
   count(Term, L, N).
```

Equality and comparison

- @<, @=<, @>, @>=

```
?- paul @< peter.
yes
?- f(2) @< f(3).
yes
?- g(2) @< f(3).
no
?- g(2) @>= f(3).
yes
?- f(a,g(b),c) @< f(a,h(a),a).
yes
```

Summary

- Testing the type of terms
- Constructing and decomposing terms
- Equality and comparison
- Database manipulation
- Control facilities
- *bagof*, *setof* and *findall*