CS370

Symbolic Programming
Declarative Programming

LECTURE 5: Lists, Operators, and Arithmetic

Jong C. Park

park@cs.kaist.ac.kr

Computer Science Department

Korea Advanced Institute of Science and Technology

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

Lists, Operators and Arithmetic

- Representation of lists
- Some operations on lists
- Operator notation
- Arithmetic

Representation of lists

Note

- All structured objects in Prolog are trees.

Example lists

- [anna, tennis, tom, skiing]
- .(anna, .(tennis, .(tom, .(skiing, [ ]))))

```
?- List1 = [a,b,c], List2 = .(a, .(b, .(c, [ ]))).
List1 = [a,b,c], List2 = [a,b,c].
```

Representation of lists

- **A list is a data structure that is either empty or consists of two parts: a head and a tail.**
  - The tail itself has to be a list.
- **Lists are handled in Prolog as a special case of binary trees.**

Some operations on lists

- Membership
- Concatenation
- Adding an item
- Deleting an item
- Sublist
- Permutations
- Problems

Some operations on lists

⊙ MEMBERSHIP

- The membership relation

  member(X,L)

- Intended behavior

```
?- member(b,[a,b,c]).
yes
?- member(b,[a,[b,c]]).
no
?- member([b,c],[a,[b,c]]).
yes
```

Some operations on lists

- Observation
  - X is a member of L if either:
    - X is the head of L, or
    - X is a member of the tail of L.
- Sample program

```
member(X,[X|Tail]).
member(X,[Head|Tail]) :-
        member(X,Tail).
```

Some operations on lists

⊙CONCATENATION

- The concatenation relation

  conc(L1,L2,L3)

- Intended behavior

```
?- conc([a,b],[c,d],[a,b,c,d]).
yes
?- conc([a,b],[c,d],[a,b,a,c,d]).
no
```

Some operations on lists

- Observation
  - If the first argument is the empty list then the second and the third arguments must be the same list.

```
conc([ ], L, L).
```

  - Otherwise the first argument has a head and a tail and must look like [X|L1].

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

Some operations on lists

- Decomposition
  - We can use the conc program to decompose a given list into two lists.

```
?- conc(L1,L2,[a,b,c]).
L1 = [ ]
L2 = [a,b,c];
L1 = [a]
L2 = [b,c];
L1 = [a,b]
L2 = [c];
L1 = [a,b,c]
L2 = [ ]
```

Some operations on lists

Pattern matching

- We can use the program to look for a certain pattern in a list.

```
?- conc(Before,[may|After],
    [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec]).
Before = [jan,feb,mar,apr]
After = [jun,jul,aug,sep,oct,nov,dec]
?- conc(_,[Month1,may,Month2|_],
        [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec]).
Month1 = apr
Month2 = jun
```

Some operations on lists

- **What are these for?**

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],
   conc(L2,[z,z,z|_],L1).
```

Some operations on lists

⊙ADDING AN ITEM

- Very simple!

```
add(X,L,[X|L]).
```

Some operations on lists

⊙ DELETION

- The deletion relation

  del(X,L,L1)

- Sample program

```
del(X,[X|Tail],Tail).
del(X,[Y|Tail],[Y|Tail1]) :-
  del(X,Tail,Tail1).
```

Some operations on lists

- The deletion program is non-deterministic.

```
?- del(a,[a,b,a,a],L).
L = [b,a,a];
L = [a,b,a];
L = [a,b,a];
no
```

Some operations on lists

- The deletion program can be used in the inverse direction.

```
?- del(a,L,[1,2,3]).
L = [a,1,2,3];
L = [1,a,2,3];
L = [1,2,a,3];
L = [1,2,3,a];
no
```

Some operations on lists

⊙SUBLIST

- Intended behavior

```
?- sublist([c,d,e],[a,b,c,d,e,f]).
yes
?- sublist([c,e],[a,b,c,d,e,f]).
no
```

- Sample program

```
sublist(S,L) :-
  conc(L1,L2,L),
  conc(S,L3,L2).
```

Some operations on lists

- The sublist relation can be used to find all sublists of a given list.

```
?- sublist(S,[a,b,c]).
S = [ ];
S = [a];
S = [a,b];
S = [a,b,c];
S = [ ];
...
```

Some operations on lists

⊙ PERMUTATIONS

- Intended behavior

```
?- permutation([a,b,c],P).
P = [a,b,c];
P = [a,c,b];
P = [b,a,c];
...
```

Some operations on lists

- Observation
  - If the first list is empty then the second list must also be empty.
  - If the first list is not empty then it has the form [X|L], and a permutation of such a list can be constructed by first permuting L for L1 and then inserting X at any position into L1.

Some operations on lists

- Sample program

```
permutation([ ],[ ]).
permutation([X|L],P) :-
  permutation(L,L1),
  insert(X,L1,P).
```

Problems

⊙Complete the following programs

```
last(Item,List) :-


reverse([ ], [ ]).
reverse([First|Rest],Reversed) :-
```

There are many other ways to reverse a given list.

Operator notation

Motivation

- Example
  - 2*a+b*c
  - Is it +(*(2,a),*(b,c)) or *(*(2,+(a,b)),c)?

Precedence

- The operator with the highest precedence is understood as the principal functor of the term.
- Which is higher: + or *?

Operator notation

- **A programmer can define his or her own operators.**

  peter has information.

  floor supports table.

  has(peter,information).

  supports(floor,table).

Operator notation

Directives

- :- op(600,xfx,has).
  - 'has' is defined as an operator.
  - its precedence is 600.
  - its type is 'xfx', a kind of infix operator.
- The operator names are atoms.
- The range is fixed, e.g. between 1 and 1200.

Operator notation

⊙ Operator types

- infix operators of three types
  - xfx, xfy, yfx
- prefix operators of two types
  - fx, fy
- postfix operators of two types
  - xf, yf

Operator notation

Precedence of argument

- If an argument is enclosed in parentheses or it is an unstructured object then its precedence is 0.
- If an argument is a structure then its precedence is equal to the precedence of its principal functor.

Operator notation

- **'x' and 'y'**
  - 'x' represents an argument whose precedence must be strictly lower than that of the operator.
  - 'y' represents an argument whose precedence is lower or equal to that of the operator.
- **Example**
  - What is the type of '-'?
  - Is it a - b - c as (a - b) - c, or as a - (b - c)?

Operator notation

Another example

- What is the type of ‘not’?
  - Is not not p allowed?

Predefined operators

- Figure 3.8

Example

- ~(A&B) <====> ~A v ~B

Arithmetic

- **A subset of the predefined operators can be used for basic arithmetic operations.**
  - +, -, *, /, **, //, mod

```
?- X = 1 + 2.
X = 1 + 2.
?- X is 1 + 2.
X = 3.
```

Arithmetic

- **Arithmetic is also used for comparison.**

```
?- 277*37 > 10000.
yes
?- born(Name,Year),
   Year >= 1980,
   Year =< 1990.
```

>, <, >=, =<, =:=, =\=

Arithmetic

- **Sample interaction**

```
?- 1 + 2 =:= 2 + 1.
yes
?- 1 + 2 = 2 + 1.
no
?- 1 + A = B + 2.
A = 2
B = 1
```

Arithmetic

⊙Example use of arithmetic operations

- Greatest Common Divisor (GCD)
  - If X and Y are equal then D is equal to X.
  - If X<Y then D is equal to the gcd of X and the difference Y-X.
  - If Y<X then do the same as above with X and Y interchanged.

```
gcd(X,X,X).
gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).
gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

Arithmetic

⊙Example use of arithmetic operations

- Counting items in a list: length(List,N)
  - If the list is empty then its length is 0.
  - If the list is not empty then List = [Head|Tail]; so its length is equal to 1 plus the length of Tail.

```
length([ ], 0).
length([_|Tail],N) :-
        length(Tail,N1),
        N is 1+N1.
```

Summary

- **List**
  - Head, Tail
- **Common operations on lists**
  - list membership
  - concatenation
  - adding an item
  - deleting an item
  - sublist
- **Operator notation**
  - infix, prefix and suffix operators
  - precedence of an operator