UNIT-IV: Declarative Programming Paradigm: Logic Programming

Faculty In-charge

Mrinmoyee Mukherjee
Assistant Professor (IT Dept.)
email: mrinmoyeemukherjee@sfit.ac.in
Mob: 9324378409

Academic Year: 2023-24

4.2: Lists and Arithmetic Execution Order

Prolog Symbols and Operators

English	Prolog
If	:-
Not	Not
Or (disjunction)	;
And (conjunction)	,

Operator	Meaning
X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater than or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	the X and Y values are equal
X =\= Y	the X and Y values are not equal

LIST

1. In Prolog LIST refers to an ordered sequence of elements

2. A single element in a list can be represented as [a]

3. An empty list can be represented as []

4. The elements of lists are separated by commas

5. Compound lists are also possible

6. [first,second,third] = [A|B]

where *A = first* and

B=[second,third]

The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.

LIST continues....

- [ ] /* this is a special list, it is called the empty list because it contains nothing */

- Now lets consider some comparisons of lists:
- [a,b,c] unifies with [Head|Tail] resulting in Head=a and Tail=[b,c]

- [a] unifies with [H|T] resulting in H=a and T=[]

- [a,b,c] unifies with [a|T] resulting in T=[b,c]

- [a,b,c] doesn't unify with [b|T]

- [] doesn't unify with [H|T]

- [] unifies with []. Two empty lists always match

Operations on List

Operations	Definition
Membership Checking	During this operation, we can verify whether a given element is member of specified list or not?
Length Calculation	With this operation, we can find the length of a list.
Concatenation	Concatenation is an operation which is used to join/add two lists.
Delete Items	This operation removes the specified element from a list.
Append Items	Append operation adds one list into another (as an item).
Insert Items	This operation inserts a given item into a list.

Operations on List

- Membership Checking
  - whether a member X is present in list L or not?
  - Consider the predicate name is list_member(X, L).
- To design this predicate, we can write program by checking
- X is a member of L if either −
  - X is head of L, or
  - X is a member of the tail of L
- The prolog program

```
list_member(X, [X| _ ]).
list_member(X, [ _ | TAIL]) :- list_member(X, TAIL).
```

?- list_member(b, [a, b, c]).	Yes
?- list_member(b, [a, [b,c]]).	?
?- list_member([b, c], [a, [b,c]]).	Yes
?- list_member(d, [a,b,c]).	No

LIST continues....

```
?- X=23,Y=24,Z=[alpha,beta],write('List is: '),write([X,Y,Z]),nl.
List is: [23,24,[alpha,beta]]
X = 23,
Y = 24,
Z = [alpha, beta].

?-
```

- Write([1,2,3])
- member(x, [x,y,z]).
- member(p, [x,y,z]).
- member(my(x,y,z),[q,r,s,my(x,y,z),w]).
- member(v,[]).
- length([a,b],M).
- length([1,2,3],M).
- length([1,2,3],a).
- length([1,2,3],X1).
- length([1,2,3],X-1).
- length([[a,c],[e,f],[h,i]],N).
- length([],P).
- length([a,b,c],3).

- reverse([1,2,3],A).
- reverse(B, [1,2,3]).
- reverse([[dog,cat],[1,2],[bird,mouse]],L).
- reverse([1,2,3,4],[4,3,6,8]).
- reverse([1,2,3,4],[4,3,2,1]).
- append([],[1,2,3],L).
- append([a,b],[1,2,3],L).
- append([a,b,23],[1,2,3],L).

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
?- write([1,2,3]).
[1,2,3]
true.

?- member(x,[x,y,z]).
true .

?- member(p,[x,y,z]).
false.

?- member(my(x,y,z),[q,r,s,my(x,y,z),w]).
true .

?- member(my(x,y,p),[q,r,s,my(x,y,z),w]).
false.

?- member(v,[]).
false.

?-
```

Type here to search

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
?- length([a,b],M).
M = 2.

?- length([1,2,3],M).
M = 3.

?- length([1,2,3],M-1).
ERROR: Type error: `integer' expected, found `_9032-1' (a
compound)
ERROR: In:
ERROR:   [11] throw(error(type_error(integer,...),context(
...,_9096)))
ERROR:    [9] toplevel_call('<garbage_collected>') at c:/p
rogram files/swipl/boot/toplevel.pl:1158
ERROR:
ERROR: Note: some frames are missing due to last-call op
mization.
ERROR: Re-run your program in debug mode (:- debug.) tc
t more detail.
```

Slide 42 of 52

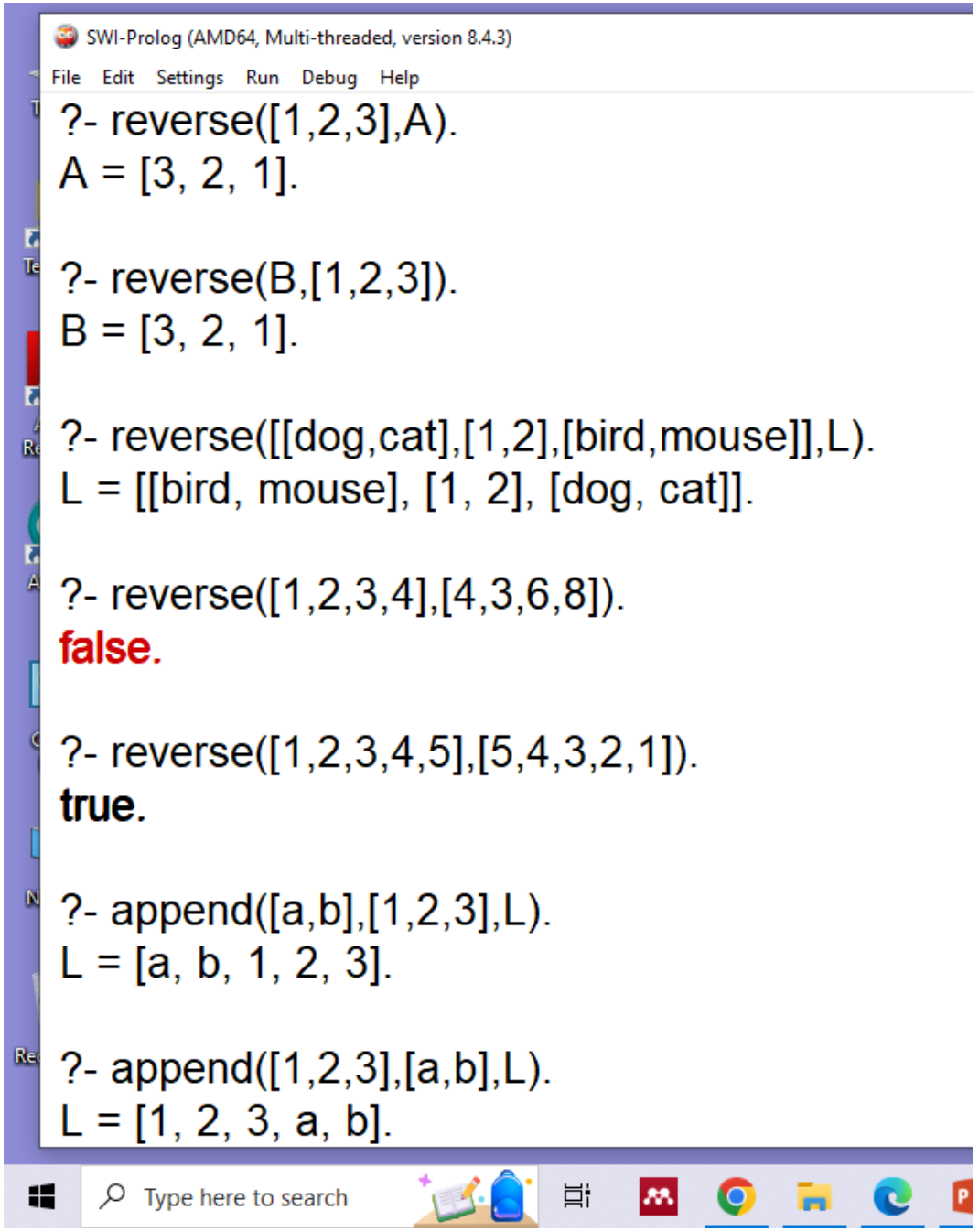
Type here to search

```
?- length([[a,c],[e,f],[h,i]],N).
N = 3.

?- length([],N).
N = 0.

?- length([[],[],[],[]],N).
N = 4.

?- length([a,b,c],3).
true.
```



Length of List

- Length Calculation:
- lengthList(L,N).
- If list is empty, then length is 0.
- If the list is not empty, then L = [Head|Tail], then its length is 1 + length of Tail.
- The prolog program

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

- EX.

```
?-lengthList([a,b,c,d,e,f,g,h,i,j], Len).
Len=10
?-lengthList([[a,b],[c,d],[e,f]],Len).
Len=3
```

Concatenation of Lists

- Concatenation
- If the first list is empty, and second list is L, then the resultant list will be L.
- If the first list is not empty, then write this as [Head|Tail], concatenate Tail with second list L2 recursively, and store into new list in the form, [Head|New List].

```
list_concat([], L, L).
list_concat([X1 | L1], L2, [X1 | L3]) :- list_concat(L1, L2, L3).
```

- EX.

```
?-list_concat([1,2], [a,b,c], NewList).
NewList = [1,2,a,b,c]
?-list_concat([[1,2,3],[p,q,r]],[a,b,c],NewList).
NewList = [1,2,3,p,q,r,a,b,c]
```

https://www.youtube.com/watch?v=syGMNm25mfY

Delete From List

- Delete element from list
  - If X is the only element, then after deleting it, it will return empty list.
  - If X is head of L, the resultant list will be the Tail part.
  - If X is present in the Tail part, then delete from there recursively.
- The Prolog Program

```
list_delete(X, [X],  []).
list_delete(X, [X | L1], [L1]).
list_delete(X, [Y | L2], [Y|L1]) :- list_delete(X, L2, L1).
```

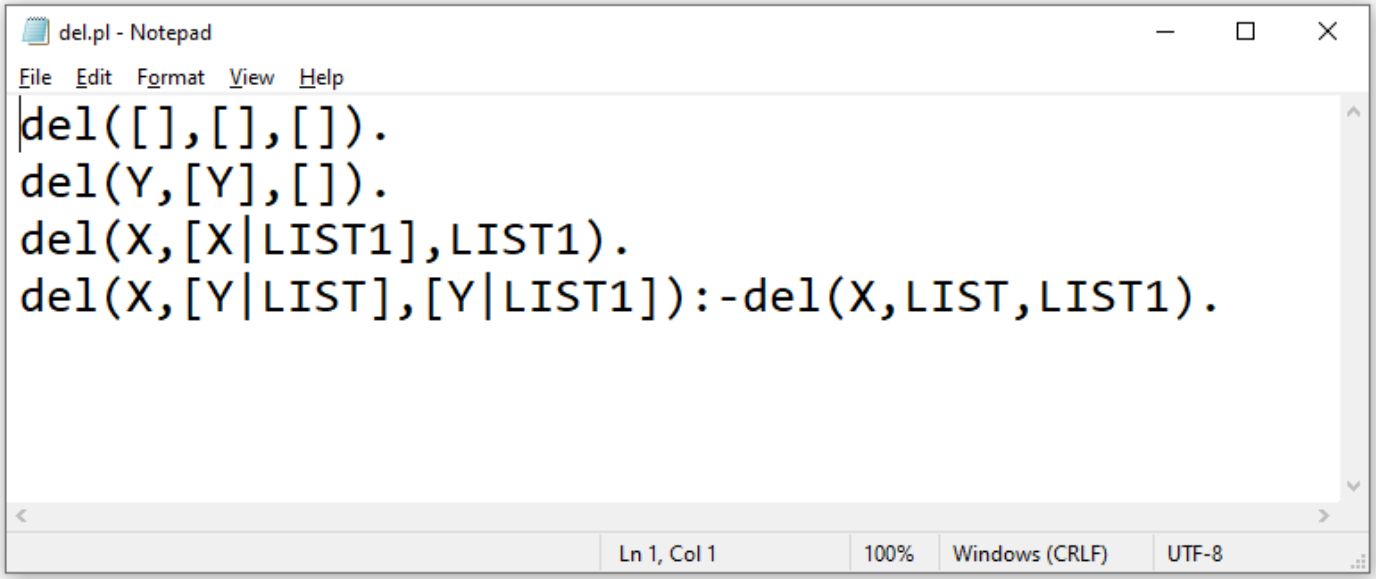
```
?- del([],[],L).
L = [] ;
false.

?- del([2],[2],L).
false.

?- del(2,[2],L).
L = [] ;
L = [] ;
false.

?- del(3,[3,4,5,6],L2).
L2 = [4, 5, 6] ;
false.

?- del(4,[1,2,3,4],X).
X = [1, 2, 3] ;
X = [1, 2, 3]
```



Arithmetic examples	**Prolog Notation**
$6+2=8$	`8 is 6+2.`
$6*2=12$	`12 is 6*2.`
$6-2=4$	`4 is 6-2.`
$6-8=-2$	`-2 is 6-8.`
$6\div 2=3$	`3 is 6/2.`
$7\div 2=3$	`3 is 7/2.`
1 is the remainder when 7 is divided by 2	`1 is mod(7,2).`

- X is 3+2.
- x is 3+2.
- 3+2=X.
- X is +(3,2).
- 5 is 3+2.
- 3+2 is 5.
- X is 3*2.
- X is 3-2.
- X is -(2,3).
- X is 5-3-1.
- X is -(5,3,1).
- X is -(-(5,3),1).
- X is 3/5.
- X is 3 mod 5.
- X is 5 mod 3.
- X is 5^3.
- X is (5^3)^2.
- X = (5^3)^2.
- 25 is 5^2.
- Y is 3+2*4-1.
- Y is (3+2)*(4)-(1).
- Y is -(*(+(3,2),4),1).
- X is 3*2, Y is X*2.
- X is abs(9).
- X is sin(90).
- X is cos(90).
- X is max(3,4).
- X is min(3,4).
- X is sqrt(49).

- 5=\=5.
- 5==5.
- 5>=5.
- 5>5.

When there is more than one operator in an arithmetic expression, e.g. **A+B*C-D**

-calculate the product of B and C

-add it to A and then subtract D

Usage of Operators

- Arithmetic calc program
- X is 100 + 200,write('100 + 200 is '),write(X),nl.
- Y is 400 - 150,write('400 - 150 is '),write(Y),nl.
- Z is 10 * 300,write('10 * 300 is '),write(Z),nl.
- A is 100 / 30,write('100 / 30 is '),write(A),nl.
- B is 100 // 30,write('100 // 30 is '),write(B),nl.
- C is 100 ** 2,write('100 ** 2 is '),write(C),nl.
- D is 100 mod 30,write('100 mod 30 is '),write(D),nl.

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
400-150 is 250
Y = 250.

?- Z is 10*300,write('10*300 is '),write(Z),nl.
10*300 is 3000
Z = 3000.

?- A is 100/30,write('10/30 is '),write(A),nl.
10/30 is 3.3333333333333335
A = 3.3333333333333335.

?- B is 100//30,write('100//30 is '),write(B),nl.
100//30 is 3
B = 3.

?- C is 100**2,write('100**2 is '),write(C),nl.
100**2 is 10000
C = 10000.

?-
```

Usage of Operators

- X+Y, the sum of X and Y
- X-Y, the difference of X and Y
- X*Y, the product of X and Y
- X/Y, the quotient of X and Y
- X^Y, X to the power of Y
- -X, the negative of X
- abs(X), the absolute value of X
- sin(X), the sine of X (for X measured in degrees)
- cos(X), the cosine of X (for X measured in degrees)
- max(X,Y), the larger of X and Y
- sqrt(X), the square root of X
- << >> Left and right shift

Database Manipulation

- Prolog has four database manipulation commands: assert, retract, asserta, and assertz.

- To assert or insert the facts in the database or knowledge-base

?- listing.

yes (It means database is empty)

- To add record to database

?- assert(happy(mia)).

yes

?- listing.

happy(mia)

?- assert(happy(vincent)).

yes

Database Manipulation

```
?- assert(happy(marcellus)).
  yes
?- assert(happy(john)).
  yes
?- assert(happy(vincent)).
  yes
?- assert ( (naive(X):- happy(X)) ).
yes
  ?- listing.
  happy(mia).
  happy(vincent).
  happy(marcellus).
  happy(john).
  happy(vincent).
  naive(A):- happy(A).
  yes
```

Database Manipulation

- To remove information predicate retract is used

```
?-retract(happy(marcellus)).
?-listing.
happy(mia).
happy(vincent).
happy(john).
happy(vincent).
naive(A)  :- happy(A).
```

- To remove all of our assertions

```
?-retract(happy(X)).
X = mia ;
X = john ;
X = vincent ;
no
 ?- listing.
naive(A)  :-  happy(A).
```

Database Manipulation

- If we want more control over where the asserted material is placed, there are two variants of assert/1 , namely:
- assertz. Places asserted material at the end of the database.
- asserta. Places asserted material at the beginning of the database.
- For example, suppose we start with an empty database, and then we give the following command:
- assert(p(b)), assertz(p(c)), asserta(p(a)).
- Then a listing reveals that we now have the following database:

```
?- listing.
p(a).
p(b).
p(c).
yes
```

END