

# COMP3411-9814- Artificial Intelligence



## Prolog Lists, Operatores & Arithmetic

---

2019 – Summer Term

Tatjana Zrimec



# SWI Prolog

---

- ◆ SWI-Prolog
- ◆ <http://www.swi-prolog.org>
- ◆ SWI-Prolog reference manual
- ◆ [http://www.swi-prolog.org/pldoc/doc\\_for?object=manual](http://www.swi-prolog.org/pldoc/doc_for?object=manual)

# SWI Prolog



SWI Prolog

Robust, mature, free. **Prolog for the real world.**

Home

DOWNLOAD

DOCUMENTATION

TUTORIALS

COMMUNITY

USERS

WIKI

*-Prolog offers a comprehensive free Prolog environment. Since its start in 1977, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. [more ...](#)*



# Outline

---

- ◆ Lists
- ◆ Operators
- ◆ Arithmetic



## List

---

- ◆ The *list* is a special structure in Prolog.,
- ◆ A list is a collection of terms, which is useful for grouping items together, or for dealing with large volumes of related data, etc.
- ◆ Examples :

```
[ a, b, c, d] %Lists are enclosed by square brackets, and items are separated by commas.  
% The length of a list is the number of items it contains. The length of this list is 4.
```

```
[ ] % Empry Lists  
[ ann, tennis, tom, running]  
[ link(a,b), link(a,c), link(b,d)]  
[ a, [b,c], d, [ ], [a,a,a], f(X,Y) ]
```



## Representation of List with Head And Tail

- ◆ We can think of a list as being made up of two parts: the first element, known as the **Head**, and everything else, called the **Tail**.
- ◆ Prolog uses a built-in operator, the pipe ( `|` ) in order to give us this split for a list.

- ◆ The list

`[red, white, black, yellow]`  
can be written as

`[Head|Tail] = [red, white, black, yellow].`

here is

**Head = red**

**Tail = [white, black, yellow]**



## Representation of List with Head And Tail

---

### ◆ Lists

$L = [\text{Head}|\text{Tail}]$

$L = [a, b, c] = [a \mid [b, c]] = [a, b \mid [c]] = [a, b, c \mid []]$

### ◆ In Prolog

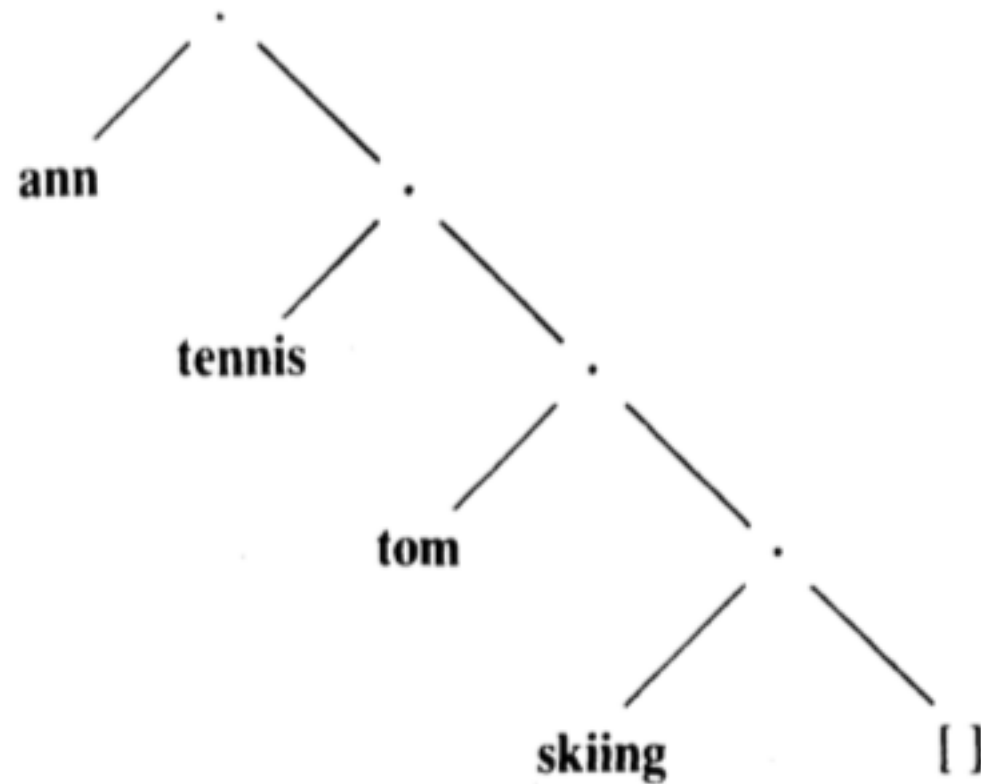
$.( \text{Head}, \text{Tail} )$

$[a,b,c] = .(a,.(b,.(c,[])))$



## Tree representation of a List

---



[ann, tennis, tom, skiing]





## Representation of List

**?- Hobbies1 = .( tennis, .( music, [] ) ),  
Hobbies2 = [ skiing, food],  
L = [ ann, Hobbies1, tom, Hobbies2].**

**Hobbies1 = [ tennis, music]  
Hobbies2 = [ skiing, food]  
L = [ ann, [tennis,music], tom, [skiing,food] ]**

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

**List1 = [a,b,c]  
List2 = [a,b,c]**



## Membership – member/2

---

`% member( X, L): X is member of L`

`member( X, [ X | _]).`      `% X appears as head of list`

`member( X, [ _ | L]) :-`  
    `member( X, L).`      `% X in tail of list`



## Different usage of member/2

---

```
?- member( c, [a,b,c,d]).      % Is an element of a given list
yes
?- member( X, [a,b,c,d]).      % Search for any element in the list
X = a;
X = b;
...
?- member( a, L).              % Find a List L containing "a"
L = [a | _] ;
L = [_ , a | _] ;              % "a" is the first element in L
L = [_ , _ , a | _] ;          % "a" is the second element in L
...
```



## Concatenation Of Lists

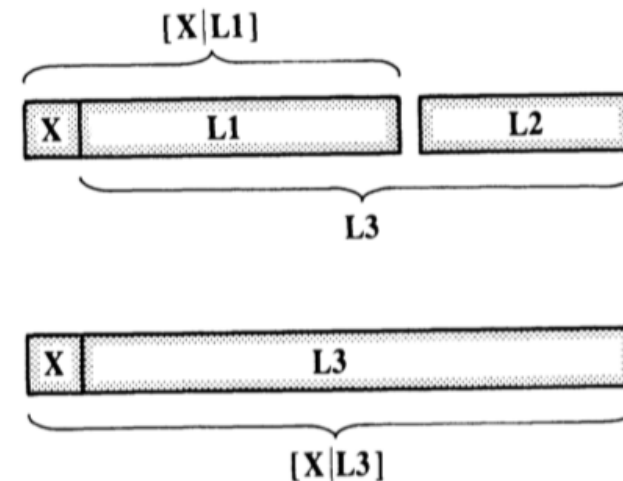
% conc( L1, L2, L3): L3 is concatenation of L1 and L2

conc( [ ], L, L).

% Base case

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

% Recursive case





## Examples of using Conc

---

?- conc( [a,b,c], [1,2,3], L).

L = [a,b,c,1,2,3]

?- conc( [a,[b,c],d], [a,[ ],b], L).

L = [a, [b,c], d, a, [ ], b]

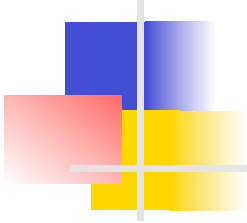
?- conc( L1, L2, [a,b,c] ).

L1 = [], L2 = [a,b,c];

L1 = [a], L2 = [b,c];

L1 = [a,b], L2 = [c];

AI-Prolog L1 = [a,b,c], L2 = []



## Example: Which months are before May, which after?

---

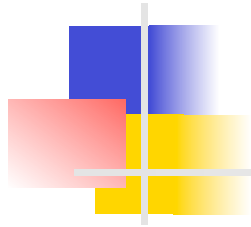
### Try in Prolog

Write a Prolog Query: Which months are before May, which after?

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

```
Before = [jan, feb,mar,apr],
```

```
After = [jun,jul,aug,sep,oct,nov,dec]
```



## Exercise

---

Write a Prolog Query: Delete from the list L1 everything from three 'z' onwards.

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],           % A list L1 with a pattern  
   conc(L2,[z,z,z|_],L1).                 % L2 is L1 up to the pattern
```



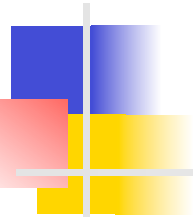
## Member by using Conc

---

- ◆ % member2( X, L): X is member of list L

```
member2( X, L ) :-  
    conc( _, [X | _ ], L ).
```





## Deleting and inserting elements in a List

---

Delete element from a List

```
% del( X, L, NewL)
```

Insertion is the reverse operation of deletion

```
% insert( X, L, NewL)
```

You can also use the "del" to insert elements in a list.



% sublist( List, Sublist): Sublist appears as a sublist  
% (subsection) in a List

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



## Definition of a List

---

```
list( []).
```

```
list( [ _ | Tail] ) :-  
    list( Tail).
```

```
% Generate lists of incremental lengths
```

```
?- list( L).
```

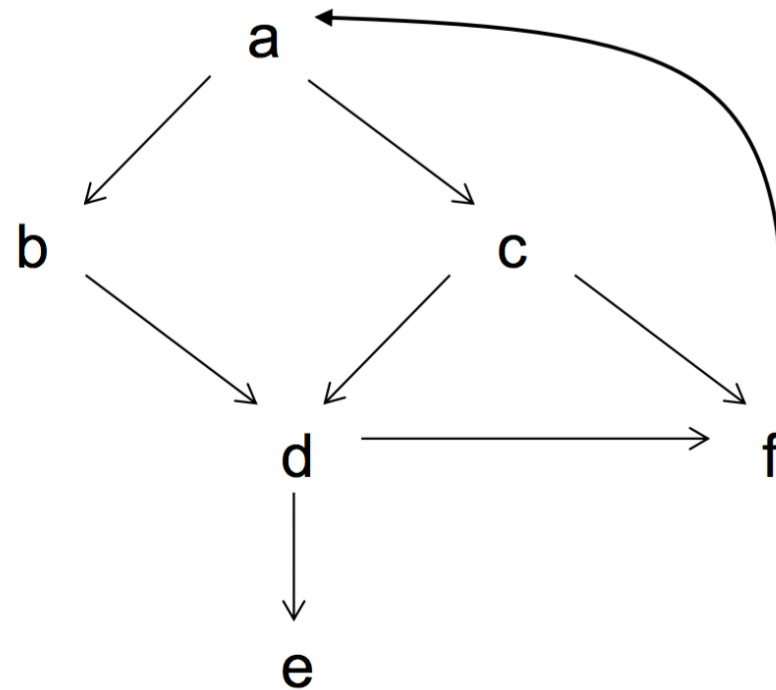
```
L = [ ];
```

```
L = [ _A];
```

```
...
```

# Path in a Graph

<b>link( a, b).</b>	<b>link( a, c).</b>
<b>link( b, d).</b>	<b>link( c, d).</b>
<b>link( c, f).</b>	<b>link( d, e).</b>
<b>link( d, f).</b>	<b>link( f, a).</b>





## Path in a Graph

---

`% path( StartNode, GoalNode): path exists between  
the nodes`

`path( Node, Node).`

`path( Start, End) :-  
 link( Start, Next),  
 path( Next, End).`



## Path in a Graph

---

```
% path( Start, Goal, Path):  
% Path = list of nodes from Start to Goal
```

```
path( Start, Start, [Start]).
```

```
path( Start, Goal, [Start | Rest]) :-  
    link( Start, Next),  
    path( Next, Goal, Rest).
```



## Exercise

---

Try:

```
?- path( a, e, P).
```

```
?- path( a, c, P).    % Problem: the search in depth misses the solution
```



## Operator notation

---

For example:

$2*a + b*c$       %only for ease of use

With operator notation:

$+( *(2,a), *(b,c) ) = 2*a + b*c$

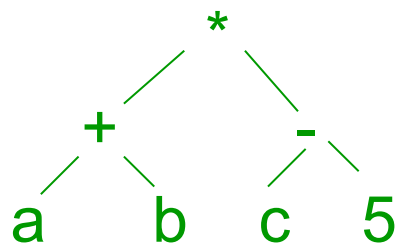
$+$ ,  $*$  are infix operators in Prolog



# The Arithmetic Expressions are also Trees

- ◆ For example:  $(a + b) * (c - 5)$
- ◆ Written as an expression with the functors:

$*(+(a, b), -(c, 5))$





## Operators in Prolog

---

- ◆ In addition to providing a user friendly operator notation for certain functors, Prolog also let's you define your own operators.

`:- op(Precedence, Type, Name).`

- ◆ Precedence is a number between 0 and 1200.

For example,

- the precedence of “=” is 700,
- the precedence of “+” is 500,
- the precedence of “\* “ is 400.



# Operators in Prolog

`:- op(Precedence, Type, Name).`

- ◆ **Type** is an atom specifying the type and associativity of the operator.
  - In the case of + this atom is **yfx**, which says that + is an infix operator f represents the operator and x and y the arguments.
  - x stands for an argument which has a precedence which is lower than the precedence of + and y stands for an argument which has a precedence which is lower or equal to the precedence of +.
  - There are the following possibilities for what Type may look like

```
infix  xfx, xfy, yfx
prefix fx, fy
suffix xf, yf
```



# Operators in Prolog

- ◆ Here are the definitions for some of the built-in operators.
- ◆ Operators with the same properties can be specified in one statement by giving a list of their names instead of a single name as third argument of `op`.

```
:- op( 1200, xfx, [ :-, --> ] ).
:- op( 1200,  fx, [ :-, ?- ] ).
:- op( 1100, xfy, [ ; ] ).
:- op( 1000, xfy, [ ', ' ] ).
:- op(  700, xfx, [ =, is, =.., ==, \==,
                  ==, =\=, <, >, =<, >= ] ).
:- op(  500, yfx, [ +, - ] ).
:- op(  500,  fx, [ +, - ] ).
:- op(  300, xfx, [ mod ] ).
:- op(  200, xfy, [ ^ ] ).
```

One final thing to note is, that operator definitions don't specify the meaning of an operator, but only describe how it can be used syntactically. An operator definition doesn't say anything about when a query involving this operator will evaluate to true. It is only a definition extending the syntax of Prolog (Brarko, pp74).



## User defined operators

---

An example how we can define two infix operators `has` and `supports` and then Prolog would allow to write “peter has information” and “floor supports table” as a facts in the database.

```
has( peter, information).  
supports( floor, table).
```

Can be written with operators:

```
:- op( 600, xfx, has).  
:- op( 600, xfx, supports).
```

```
peter has information.  
floor supports table.
```



# Arithmetic in Prolog

---



## Built-in Predicate for Arithmetic Operations

---

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

```
?- X is 1 + 2.           % “is”: built-in predicate that forces  
                          evaluation  
X=3
```

A special predefined operator **is** to invoke arithmetic!



# Arithmetic Operations

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $**$  for real numbers

$//$ ,  $\text{mod}$  for integer

$\sin$ ,  $\cos$ ,  $\log$ , ... standard functions

?- X is  $2 + \sin(3.14/2)$ .

X = 2.9999996829318345

?- A is  $11/3$ .

A = 3.6666666666666665

?- B is  $11//3$ .

B=3

?-C is  $11\text{mod}3$ .

C=2





## Comparison operators

$X > Y$	X is greater than Y
$X < Y$	X is less than Y
$X \geq Y$	X is greater than or equal to Y
$X \leq Y$	X is less than or equal to Y
$X ::= Y$	the values of X and Y are equal
$X \neq Y$	the values of X and Y are not equal

?-  $315*3 \geq 250*4$ .

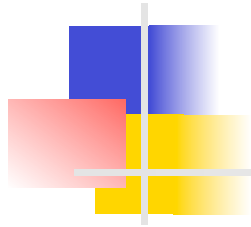
no

?-  $2+5 = 5+2$ .

no

?-  $2+5 ::= 5+2$ .

yes



## Comparison operators

---

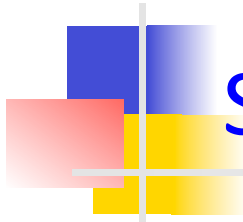
Operator

`::=`

we have to distinguish from the operator

`=`

which serves to compare the non- arithmetic terms



## Strings

---

- ◆ Strings are lists of positive integers
  - Positive integers correspond to an ASCII code character.

Prolog does not make difference between

"Prolog"    and    [80,114,111,108,111,103]



## The procedure

`name (Atom, List)`

allows the conversion of the atom into the list of ASCII code characters

?- `name(A, [112,114,111,108,111,103]).`

`A = prolog`      % No spaces without single quotation marks