U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DEPARTMENT OF INFORMATICS ENGINEERING

U.PORTO
FC FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

DEPARTMENT OF COMPUTER SCIENCE

Functional and Logic Programming

Bachelor in Informatics and Computing Engineering
2025/2026 – 1st Semester

Lists

Jácome Cunha
jacome@fe.up.pt

Daniel Castro Silva
dcs@fe.up.pt

Agenda

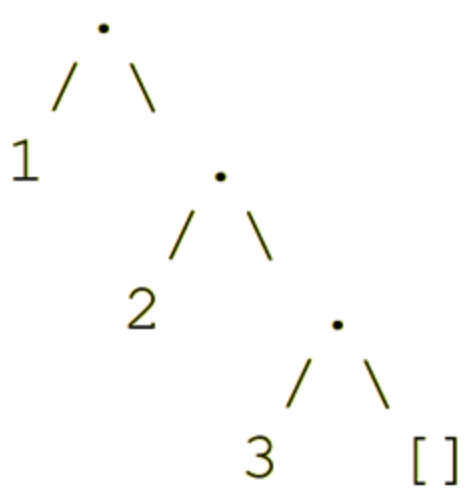
- Lists

Lists

- Lists are the quintessential data structure in Prolog
- Empty list represented as [ ]
- Elements separated by commas within square brackets
  - [a, b, c]
  - [4, 8, 15, 16, 23, 42]
- Lists elements can be anything, including other lists
  - [1, [a, b, v], g, [2, [D, y], 3], 4]

Lists

- The internal representation uses the . functor and two arguments – the head and tail of the list
  - Ex.: [1, 2, 3] = .(1, .(2, .(3, []))).




```
| ?- A = .(1, .(2, .(3, []) ) ).
A = [1,2,3] ?
yes
```

- Strings are a representation of lists of character ASCII codes

```
| ?- A = "Hello".
A = [72,101,108,108,111] ?
yes
```

Lists

- Easily separate the head of the list from the rest of the list
  - The head of the list can separate more than one element

```
[ H | T ] % where T is a list with the remaining elements of the list
[ 4 ] = [ 4 | [ ] ]          % tail of list with one element is empty list
[4, 8, 15, 16, 23, 42] = [4 | [8, 15, 16, 23, 42] ]
[4, 8, 15, 16, 23, 42] = [ 4, 8 | [ 15, 16, 23, 42] ]
```

- Definition of what is a list
  - An empty list
  - A list construct where tail is a list

```
is_list( [ ] ).
is_list( [H|T] ):- is_list(T).
```

wooclap

List Length

- There are several useful built-in predicates to work with lists
  - ***length(?List, ?Size)***
    - Size of a list (very flexible)

Can also be easily implemented recursively

```
length( [ ], 0 ).
length( [_|T], L ):-
        length(T, L1),
        L is L1+1.
```

```
| ?- length([1,2,3], 3).
yes
| ?- length([1,2,3], L).
L = 3 ?
yes
| ?- length(L, 3).
L = [_A,_B,_C] ?
yes
| ?- length(L, S).
L = [],
S = 0 ? ;
L = [_A],
S = 1 ? ;
L = [_A,_B],
S = 2 ? ;
L = [_A,_B,_C],
S = 3 ?
yes
```

List Membership

- ***member(?Elem, ?List)***
  - List member (very flexible)
- ***memberchk(?Elem, ?List)***
  - Member verification (deterministic, no backtracking because of `X \= Y`)

Can also be easily implemented recursively

```
member( X, [X|_] ).
member( X, [_|T] ):-
        member(X, T).
```

```
memberchk( X, [X|_] ).
memberchk( X, [Y|T] ):-
        X \= Y,
        memberchk(X, T).
```

```
| ?- member(2, [1,2,3]).
true ?
yes
| ?- member(2, L).
L = [2|_A] ? ;
L = [_A,2|_B] ?
yes
| ?- member(M, [1,2]).
M = 1 ? ;
M = 2 ? ;
no
| ?- member(M, L).
L = [M|_A] ? ;
L = [_A,M|_B] ?
yes
```

Appending Lists

- ***append(?L1, ?L2, ?L3)***
  - Appends two lists into a third (very flexible)

Can also be easily implemented recursively

```
append( [ ], L2, L2 ).
append( [H|T], L2, [H|T3] ):-
        append(T, L2, T3).
```

```
| ?- append([1,2], [3,4], [1,2,3,4]).
yes
| ?- append([1,2], [3,4], L).
L = [1,2,3,4] ?
yes
| ?- append([1,2], L, [1,2,3,4]).
L = [3,4] ?
yes
| ?- append(L, [3,4], [1,2,3,4]).
L = [1,2] ?
yes
| ?- append(L1, L2, [1,2,3]).
L1 = [],
L2 = [1,2,3] ? ;
L1 = [1],
L2 = [2,3] ? ;
L1 = [1,2],
L2 = [3] ? ;
L1 = [1,2,3],
L2 = [] ? ;
no
```

Sorting Lists

- ***sort(+List, -SortedList)***
  - Sorts a (proper) list
- ***keysort(+PairList, -SortedList)***
  - Sorts a (proper) key-value pair list
  - If a key appears more than once, elements retain original order

```
| ?- sort([4,2,3,1], [1,2,3,4]).
yes
| ?- sort([4,2,3,1], SL).
SL = [1,2,3,4] ?
yes
| ?- keysort([2-1, 1-2, 4-3, 3-4], SL).
SL = [1-2,2-1,3-4,4-3] ?
yes
| ?- keysort([2-1, 1-2, 4-3, 3-4, 1-1], SL).
SL = [1-2,1-1,2-1,3-4,4-3] ?
yes
```

Can also be implemented recursively
Homework!

wooclap

Lists Library

- The Lists library has numerous predicates to work with lists

- Libraries can be imported using the *use_module* directive:

```
:-use_module(library(lists)).
```

See section 10.25 of the SICStus Manual for a complete description of available predicates

You

- Choose any slide (add a letter to the slide so the others know it's already chosen)
- You don't need to identify yourself, can be anonymous
- Learn about the predicate(s) in that slide (documentation, testing it, LLM, google, etc.)
- Explain in 1 or 2 slides the predicate
  - Make sure what you write is correct
- You have 15 minutes
- In the end, you can choose to present your predicate (preferable) or leave it to the professor

Lists Library

- Some useful predicates from the lists library
  - nth0(?Pos, ?List, ?Elem) / nth1(?Pos, ?List, ?Elem)
  - nth0(?Pos, ?List, ?Elem, ?Rest) / nth1(?Pos, ?List, ?Elem, ?Rest)

```
| ?- nth1(3, R, c, [a,b,d]).
R = [a,b,c,d] ?
yes
| ?- nth1(3, [a,b,c,d], X, R),
     nth1(3, Res, e, R).
X = c,
R = [a,b,d],
Res = [a,b,e,d] ?
yes
```

```
| ?- nth1(3, [a,b,c,d], X).
X = c ?
yes
| ?- nth1(3, [a,b,c,d], X, R).
X = c,
R = [a,b,d] ?
yes
```

Can be used to remove, insert or replace (when used twice) list elements

nth0 and nth1

Predicate	Indexing starts at	Example
`nth0(Index, List, Elem)`	0 (zero-based)	`nth0(0, [a,b,c], X). % X = a`
`nth1(Index, List, Elem)`	1 (one-based)	`nth1(1, [a,b,c], X). % X = a`

Form	Meaning	Example
`nth0(I, L, E)`	Get element E at position I	`nth0(2,[a,b,c,d],X). % X=c`
`nth1(I, L, E)`	Same using 1-based index	`nth1(4,[a,b,c,d],X). % X=d`
`nth0(I, L, E)` with E known	Find index I	`nth0(I,[a,b,c],b). % I=1`
`nth0(Index, List, Elem, Rest)`	Get element and list without it (delete elem.)	`nth0(2,[a,b,c,d],X,R).` `% X=c, R=[a,b,d]`
`nth0(Index, List, Elem, Rest)`	Add an element	`nth0(2,X,c,[a,b,d]).` `% X=[a,b,c,d]`
`nth0(Index, List, Elem, Rest)`	Replace an element	`nth0(1,[a,b,c],X,R),nth0(1,S,f,R).` `% X=b, R=[a,c], S=[a,f,c]`

Lists Library

- select(?X, ?XList, ?Y, ?YList)
  - finds an occurrence of X in XList, replaces it with Y, and produces YList

- delete(+List, +ToDel, -R)
- delete(+List, +ToDel, +Count, -R)
  - Deletes Count occurrences of ToDel in List, result R

- last(?Init, ?Last, ?List)
  - Last element of List and the rest in Init

```
| ?- select(g, [a,g,a,g,a], r, X).
X = [a,r,a,g,a] ? ;
X = [a,g,a,r,a] ? ;
no
| ?- delete([a,b,b,a], a, X).
X = [b,b] ?
yes
| ?- delete([a,b,b,a], a, 1, X).
X = [b,b,a] ? ;
no
| ?- last(I, L, [1,2,3,4]).
I = [1,2,3],
L = 4 ?
yes
```

Lists Library

- segment(?List, ?Segment)
  - succeed when Segment is a contiguous subsequence of List.
- sublist(+List, ?Part, ?Before, ?Length, ?After)
  - extract a contiguous Part of List with Length size and Before/After pre/suffix

```
| ?- segment([a,b,c], S).
S = [a] ? ;
S = [a,b] ? ;
S = [a,b,c] ? ;
S = [b] ? ;
S = [b,c] ? ;
S = [c] ? ;
S = [] ? ;
no
```

```
| ?- sublist([a,b,c], Part, Bef, Len, Aft).
Part = [],
Bef = 0,
Len = 0,
Aft = 3 ? ;
Part = [a],
Bef = 0,
Len = 1,
Aft = 2 ? ;
Part = [a,b],
Bef = 0,
Len = 2,
Aft = 1 ? ;
```

```
| ?- sublist([a,b,c], S, _, _, _).
S = [] ? ;
S = [a] ? ;
S = [a,b] ? ;
S = [a,b,c] ? ;
S = [] ? ;
S = [b] ? ;
S = [b,c] ? ;
S = [] ? ;
S = [c] ? ;
S = [] ? ;
no
```

Lists Library

- append(+ListOfLists, -List)
  - concate of Haskell

```
| ?- append([[1,2,3], [4,5,6], [7,8,9]], L).
L = [1,2,3,4,5,6,7,8,9] ? ;
no
```

- reverse(?List, ?Reversed)
- rotate_list(+Amount, ?List, ?Rotated)
  - cyclically shifts (rotates) List by Amount number of positions

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

```
| ?- rotate_list(1, [a,b,c,d], L).
L = [b,c,d,a] ? ;
no
| ?- rotate_list(1, L, [a,b,c,d]).
L = [d,a,b,c] ? ;
no
```

Lists Library

- transpose(?Matrix, ?Transposed)
  - converts rows into columns (and vice-versa)
- remove_dups(+List, ?PrunedList)
- permutation(?List, ?Permutation)
  - List permutations, with backtracking

```
| ?- transpose([[1,2,3],[4,5,6],[7,8,9]],T).
T = [[1,4,7],[2,5,8],[3,6,9]] ? ;
no
| ?- remove_dups([a,b,b,a], L).
L = [a,b] ? ;
no
```

```
| ?- permutation([a,b,c], P).
P = [a,b,c] ? ;
P = [b,a,c] ? ;
P = [b,c,a] ? ;
P = [a,c,b] ? ;
P = [c,a,b] ? ;
P = [c,b,a] ? ;
no
```

Lists Library

- sumlist(+ListOfNumbers, ?Sum)
- max_member(?Max, +List)
- min_member(?Min, +List)
- max_member(:Comp, ?Max, +List)
  - Comp is a comparison predicate of arity 2 used to compare elements
- min_member(:Comp, ?Min, +List)

```
| ?- sumlist([1,2,3,4,5], S).
S = 15 ? ;
no
| ?- max_member(Max, [4,5,3,2,6,1]).
Max = 6 ? ;
no
```

Lists Library

- maplist(:Pred, +L) / maplist(:Pr, +L1, ?L2) / maplist(:Pr, +L1, ?L2, ?L3)
  - Applies predicate to each element / map / zipWith
- map_product(:Pred, +Xs, +Ys, ?List)
  - Cartesian product

```
even(X):-
        X mod 2 =:= 0.
square(X, Y):-
        Y is X*X.
pow(X, Y, Z):-
        Z is X**Y.
```

```
| ?- maplist(even, [2,3,4,5]).
no
| ?- maplist(even, [2,4]).
yes
| ?- maplist(write, [a,b,b,a]).
abba
yes
| ?- maplist(square, [2,3,4,5], L).
L = [4,9,16,25] ? ;
no
```

```
| ?- maplist(pow,[2,3,4],[2,3,4],L).
L = [4.0,27.0,256.0] ? ;
no
| ?- map_product(pow,[2,3,4],[2,3,4],L).
L = [4.0,8.0,16.0,9.0,27.0,81.0,16.0,64.0,256.0] ? ;
no
```

Lists Library

- scanlist(:Pred, +Xs, ?Start, ?Final)
  - foldl
- cumlist(:Pred, +Xs, ?Start, ?List)
  - Similar to accumulate in python

```
soma(A, B, C):-
        C is A+B.
soma2(A, B, A+B).
soma3(A, B, B+A).
```

```
| ?- cumlist(soma, [2,3,4,5], 1, F).
F = [3,6,10,15] ? ;
no
| ?- cumlist(soma2, [2,3,4,5], 1, F).
F = [2+1,3+(2+1),4+(3+(2+1)),5+(4+(3+(2+1)))] ?
yes
| ?- cumlist(soma3, [2,3,4,5], 1, F).
F = [1+2,1+2+3,1+2+3+4,1+2+3+4+5] ?
yes
```

```
| ?- scanlist(soma, [2,3,4,5], 1, F).
F = 15 ?
yes
| ?- scanlist(soma2, [2,3,4,5], 1, F).
F = 5+(4+(3+(2+1))) ?
yes
| ?- scanlist(soma3, [2,3,4,5], 1, F).
F = 1+2+3+4+5 ?
yes
```

Lists Library

```
| ?- some(even, [3,5,7]).
no
| ?- some(even, [3,4,5]).
true ?
yes
```

- some(:Pred, +List)
  - any
- include(:P, +X, ?L) / include(:P, +X, +Y, ?L) / include(:P, +X, +Y, +Z, ?L)
  - filter / P(x, y) succeeds, L ⊆ X / P(x, y, z) succeeds, L ⊆ X
- exclude(:P, +X, ?L) / exclude(:P, +X,+Y, ?L) / exclude(:P, +X,+Y,+Z, ?L)
  - not include
- group(:Pred, +List, ?Front, ?Back)
  - Group until predicate fails, splitting the list at that point

```
| ?- include(even, [1,2,3,4,5,6,7,8], L).
L = [2,4,6,8] ?
yes
| ?- exclude(even, [1,2,3,4,5,6,7,8], L).
L = [1,3,5,7] ?
yes
```

```
| ?- group(even, [2,4,6,1,2,3,4], F, B).
F = [2,4,6],
B = [1,2,3,4] ?
yes
```

Lists

- Several of these predicates can be implemented using append
  - However, sometimes we can find more efficient versions

- Example: list reverse

```
reverse([], []).
reverse([X|Xs], Rev):-
        reverse(Xs, Ys),
        append(Ys, [X], Rev).
```

Lists

- We can use an accumulator (tail recursion) to reverse the list

```
reverse(Xs, Rev):- reverse(Xs, [], Rev).
reverse([X|Xs], Acc, Rev):-
        reverse(Xs, [X|Acc], Rev).
reverse([], Rev, Rev).
```

- The accumulator holds the reversed list in the last step of the recursion

Q & A

≤ in different programing languages