

More Prolog

Hacks and features of Prolog making it into a full programming language:

- General data structures and lists
- Control facilities
- Arithmetic in Prolog
- Syntactic extensibility: Operator notation
- (Self-inspection and modification)

That's really all of it!

Basic notions, now adding *structures*

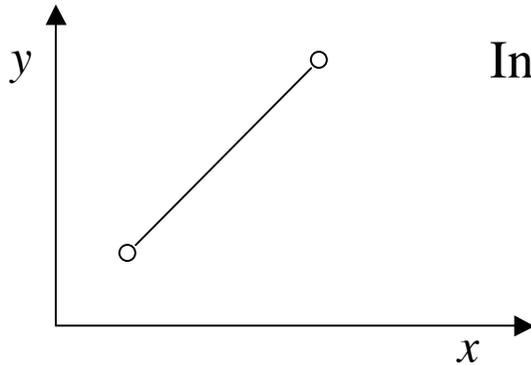
- *predicates*: **parent**
 - defines a *relation*
 - given by *facts, rules*, coll. called *clauses*
- *constants*: **tom, bob, x, y**
- *variables*: **x, y, Tom**
- *atoms*: **parent(A, a)**
- Arguments to predicates can also be *structures*:

point(1,1)

line_segment(point(1,1), point(2,2))

NB: Looks like pred's with arguments, ...

An example of using structures



Intuitive interpretation of structure:

```
line_segment(point(1,1),point(3,3))
```

How many lines of Java
is needed to produce
a similar functionality????

This is a program:

```
vertical( line_segment(point(X,Y), point(X,Y1))).  
horizontal( line_segment(point(X,Y), point(X1,Y))).
```

Querying it:

```
?- vertical(line_segment(point(1,1),point(2,Y))).  
no  
?- horizontal(line_segment(point(1,1),point(2,Y))).  
Y = 1 ?
```

Lists, an important sort of structures

List syntax \approx syntactic sugar; no new semantics

```
?- write([1,2,3,4,5,6]).
```

```
[1,2,3,4,5,6]
```

```
?- write_canonical([1,2,3,4,5,6]).
```

```
'.'(1, '.'(2, '.'(3, '.'(4, '.'(5, '.'(6, []))))))
```

```
?- [1,2,3,4,5,6] = [Head | Tail].
```

```
Head = 1, Tail = [2,3,4,5,6]
```

```
?- [First, Second | Tail2] = [a,b,c,d,e,f].
```

```
First = a, Second = b, Tail2 = [c,d,e,f]
```

Working with lists; the member predicate

A built-in predicate; in SICStus (v. 3, not 4, sic!) remember this:

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

```
?- member(a,[a,b,c]).
```

```
yes
```

Member is also a list *constructor*:

```
?- member(a,L), member(b,L), member(c,L).
```

```
L = [a,b,c|_A]
```

Implementation of member

```
member(X, [X | _] ).
```

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

"append": List concat'n & decomp'n

Examples:

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

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

```
?- append(X,Y,[a,b,c]).
```

```
X = [], Y = [a,b,c] ? ;
```

```
X = [a], Y = [b,c] ? ;
```

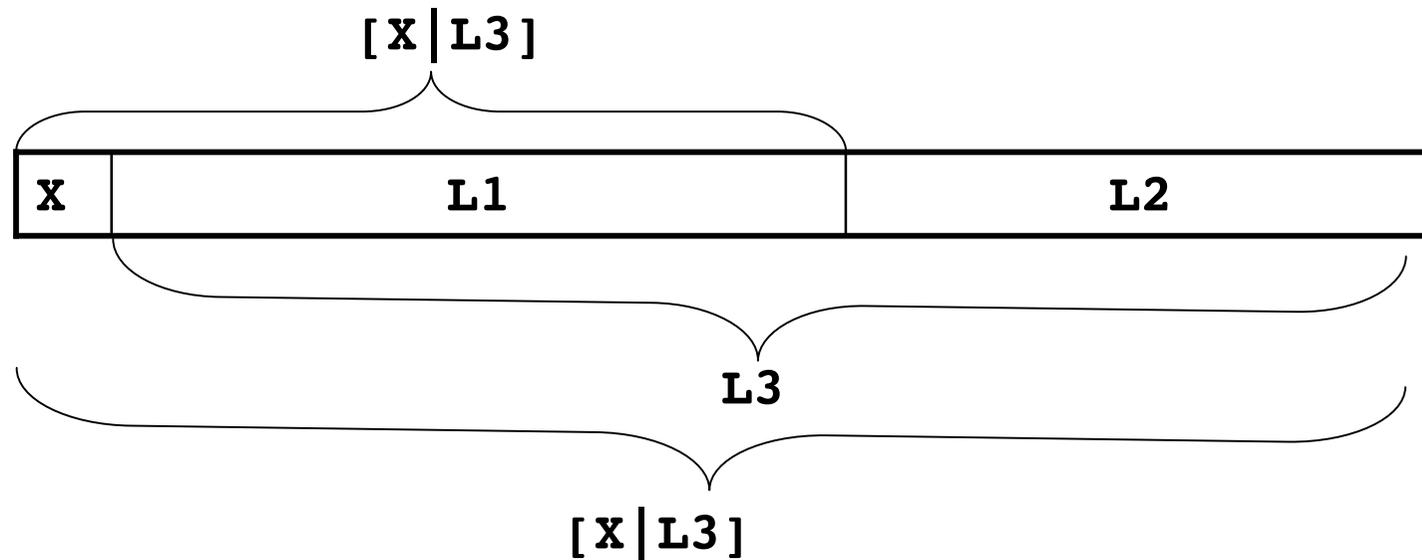
```
X = [a,b], Y = [c] ? ;
```

```
X = [a,b,c], Y = [] ? ;
```

A definition of "append"

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

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



Arithmetic, a stepchild in Prolog

```
?- X is 2 + 2 * 3.
```

```
X = 8 ?
```

```
?- X is 2 + Y * 3.
```

```
! Instantiation error in argument 2 of is/2
```

```
! goal:  _79 is 2+_73*3
```

Remember points about

- range-restrictedness
- left-to-right execution

Exercises

- 5.1, p. 46
- 5.2, p. 46–47.
 - Only triangles, **identical_triangles**, **segment_length** and possibly **area**(<triangle>, <length>)
- 5.3, p.47.
- Extra: Define, using **append**, a predicate **find_abc(L)**, which is satisfied iff **[a,b,c]** is a “sublist” of **L**, e.g.

```
?- find_abc([k,l,m,n,a,b,c,d,e])
```

```
yes
```

```
?- find_abc([k,l,m,n,a,b,z,z,c,d,e])
```

```
no
```

(can be done with just *one* call to **append**)

Useful built-ins (use with care)

... optimization for special cases

- var**(*arg*) — argument *currently* uninstantiated?
- nonvar**(*arg*) — the opposite
- ground**(*arg*) — is *current* value of *arg* ground, i.e., variable-free?
- atom**(*arg*) — *current* value constant that is not a number?
- integer**(*arg*) — *current* value an integer number?
- atomic**(*arg*) — *current* value a constant?

Splitting terms by "=.."

```
?- f(a,b) =.. [F|Args].
```

```
F = f, Args = [a,b]
```

```
?- f(a,b) =.. [F|Args], NewTerm =.. [F,new|Args].
```

```
..., NewTerm = f(new,a,b)
```

Useful for translating
one program into another...

Control of backtracking by "!" (cut)

```
salary(S, 0):- student(S), !.  
salary(S, 1000000).  
student(peter).
```

```
?- salary(peter,S).
```

```
S = 0 ;
```

```
no
```

```
?- salary(jane, S).
```

```
S = 1000000 ;
```

```
no
```

But trying to generate all solution :(

```
?- salary(X,S).
```

```
X = peter, S = 0 ;
```

```
no
```

Be careful:

- Destroys logic
- Introduces assumptions about how predicates are called

Conditionals

```
salary(X,S):-  
    student(X) -> S=0  
    ;  
    director(X) -> S=1000000  
    ;  
    professor(X) -> S=500000  
    ;  
    S = 10.
```

Like a "soft-cut", successful-test-and-choice not backtracked,
but subsequent clause may be used.

Operators: Extensible syntax

```
:- op(700, xfx, sparker).
```

```
manden sparker hunden.
```

```
:- op(700, xfx, bider).
```

```
X bider Y :- Y sparker X.
```

Important: Only syntactic sugar, no new semantics

```
?- current_op(X, Y, Z).
```

```
X = 1200, Y = xfx, Z = :- ? ;
```

```
X = 1200, Y = xfx, Z = --> ? ;
```

```
...
```

```
X = 1000, Y = xfy, Z = ', ' ? ;
```

```
...
```

```
X = 500, Y = yfx, Z = + ? ;
```

```
...
```

```
X = 400, Y = yfx, Z = * ? ;
```

Example of program with operators

`:- op(700, xfx, er).`

`:- op(100, fx, [en,et]).`

`en mand er et menneske.`

`en kvinde er et menneske.`

`et menneske er et dyr.`

`en ko er et dyr.`

`peter er en mand.`

`X er Z :- X er Y, Y er Z.`

Other facilities

Generating all solutions:

setof, bagof, findall

— read about them when you need them

Input-output:

write('Hello') useful for test prints...

Inspecting and modifying the program

clause, asserta, assertz, retract

We may see those guys
later in the course

This is really all of Prolog!

