

Introduction to PROgramming in LOGic

- What is a declarative language?
- Using relation
- Prolog is based on Horn-clauses
- Horn-clauses is a subset of first order predicate logic
- Prolog-programs are written in a declarative style, and have properties, relations and rules

Different descriptions

Find all grand children for a specific person X?

Declarative description (defines the relation): A grandchild GC is a child to GPs child.

Imperative description (explains how to find a grandchild): To find a grandchild to X, first find a child to X. Then find a child to this child.

Imperative description II: To find a grandchild to X, find first a parent to a child, then check if this parent is a child to X.

Imperative and declarative

Imperative:

```
read(person);
for i := 1 to maabarn do
  if barn[i, 1] = person then
    for j := 1 to maabarn do
      if barn[i, 1] = barn[i, 2] then
        writeln(barn[j, 2]);
      end if
    end for
  end if
end for
end for
```

Logic: $(\forall X) (\forall Y) ((\exists Z) barn(X, Z) \wedge barn(Z, Y)) \rightarrow barnbarn(X, Y)$

Prolog: barnbarn(X, Y) :- barn(X, Z), barn(Z, Y).

Declarative and imperative languages

	Imperative languages	Declarative languages
Philosophy	The users tells exactly howthe problem is solved	The user specifies what the problem is
Program	A sequence of commands	A set of statements
Example	Pascal, C, Ada, Java	Prolog, ML, Scheme, Gödel
Advantages	Fast and specialized programs	General, readable and possibly correct programs (formal proofs)

- Programs written in imperative languages force a specific execution order
- Declarative language does not say *how* something should be executed

- Describe the knowledge about the problem
- Separation between program and control structure (exceptions for efficiency reasons)

Logic Programming – important features

- Predicates fail or succeed
 - If they succeed, unbound variables are unified (and may be bound to values)
- Predicates do *not* return values:
 - Terms can only be unified with each other
 - Only arithmetic expressions are evaluated (is, =, :=)
 - No functions in Prolog!

Syntax

English	Predicate Calculus	Prolog
and	\wedge	,
or	\vee	;
only if	\leftarrow	::-
not	\neg	\+

A Prolog program

```
likes(lasse,lisa). %fact predicates
likes(lasse,mia).
likes(lasse,beer).
likes(mia,beer).
likes(lisa,fanta).
likes(lisa,lasse).
likes(lisa,mia).

friends(X,Y) :- %rule predicate
    likes(X,Y),
    likes(Y,X).
```

A Prolog session

- Prolog is an interactive language, the user enters queries after the Prolog prompt "? - " .

```
?- likes(Lasse, lisa).  
yes  
?- likes(lisa, mia).  
yes  
?- likes(Lasse, X).  
X=lisa ; %Request more answers  
X=mia ;  
X=beer ;  
no %No more answers  
?- likes(Lasse, wine).  
no %Negation as failure  
?- friends(Lasse, X).  
X=lisa ;  
no
```



9

Some important concepts in logic

Predicates encode properties and relations

- Predicates have an arity: its number of arguments
 - arity = 0 (no arguments) : A propositional statement
 - arity = 1: A property
 - arity = 2: A relation

Arguments are called terms.

- A term may be a
 - constant
 - variable
 - structure with terms as its components



10

Prolog and Horn-clauses

- Prolog only handles a subset of predicate logic
- Many predicate logic expressions can be translated into "horn-clauses"
- Prolog handles Horn-clauses
- Prolog has some extra-/meta-logical extensions
 - useful for efficiency reasons and to write programs needing higher order logics

Horn-clauses

Conclusions and conditions

- $(\forall x_1 \dots x_n) A \leftarrow B_1 \wedge \dots \wedge B_m \Leftrightarrow (\forall x_1 \dots x_n) A \vee \neg B_1 \vee \dots \vee \neg B_m$
- $(\forall x_1 \dots x_n) A$
 - No conditions — A is unconditionally true
- $(\forall x_1 \dots x_n) \neg A \Leftrightarrow (\forall x_1 \dots x_n) \perp \vee \neg A$
 - Predicate A is false — \perp is falsity

Facts: Properties and Relations

- A property fact in Prolog:

```
whisky(bowmore).           %Bowmore is a whisky
```

- Two relation facts in Prolog

```
region(bowmore, islay).   % Bowmore is from the region  
Islay  
age(bowmore, 12).        % Bowmore is 12 years old
```



13

Interpreter

- Ask what whiskeys are available

```
?- whisky(X).             % Request further whiskeys  
X=bowmore ? ;  
no % No further whiskeys
```



14

Anonymous variables

- If we want to know if we have some Whiskey, but don't care which one

```
?- whiskey(_).  
yes
```

- "`_`" is called an *anonymous* or *free* variable and is never bound to a value

Conjunctive queries

```
?- region(W,islay), age(W,A). % Do we have an Islay Whisky  
and how old is it?  
A=12,W=bowmore ?  
  
?- whiskey(W), age(W,A), A>8. % Do we have a Whisky that is  
more than 8 years old?  
A = 12, W = bowmore ? ;  
no
```

Lists in Prolog

- `[]`: an empty list
- `[1, 2, abba]`: a list with 3 elements
- Lists in lists: `[[11, 12, 13], [21, 22, 23], [31, 32, 33]]` (a 3x3 matrix (a list with 3 lists)).
- Different notations: `[First_Element | Tail_List]`
 - example: `[1 | [2, abba]]`

List notations

```
?- [a,b,c] = [Head|_].
Head = a
?- [a,b,c] = [_|Tail].
Tail = [b,c]
?- [a,b,c] = [a|[b,c]].
yes
?- [a] = [_|T].
T = [];
no
```

Test these in a Prolog interpreter!

A recursive predicate (member)

```
member(X, [X|Rest]). % X is a member if it is the first element
member(X, [F|Rest]) :- % otherwise X is in the rest of the list.
    member(X, Rest).

?- member(X, [abba, 1, 3]).
X = abba ? ;
X = 1 ? ;
X = 3 ? ;
no

?- member(1, [A, B]).
A = 1 ? ;
B = 1 ? ;
```

no

Arithmetic and other operators

- +, -, *, /, sin, cos, tan
- <, >, >=, =<, ::=, =\=
- is
- ==, \== (exact equality, not equal)
- = (unification)

Arithmetic examples

?- 2 =< 4.

yes

?- 2+2 ::= 4. % ::= calculates the expressions on both sides

yes

?- 2+2 ::= 5.

no

?- 2+2 =\= sin(0), 0 + 0 ::= sin(0).

yes

Debugging Prolog: Trace

```
? - trace, X is sin(4/5), X>2.
1 1 Call: _59 is sin(4/5) ?
1 1 Exit: 0.71735609 is sin(4/5) ?
2 1 Call: 0.71735609 > 2 ?
2 1 Fail: 0.71735609 > 2 ?
1 1 Redo: 0.71735609 is sin(4/5) ?
1 1 Fail: _59 is sin(4/5) ?
...
```

A is B.

- The expression on the right side of "is" is calculated and unified with variables (or constant) A.

```
?- Area is 3.14 * 5 * 5.
Area = 78.5
```

```
?- 4 is 4.
yes
```

==, \== (exact equality)

?- W == Q. % two different variables
no

?- 1+2 == 2+1. % two structures, equal to: % +(1,2) == +(2,1).

no

?- 1+2 == 1+2.

yes

?- X \== Y.

yes

Unification '=' the "core" of logic programming

- Two terms can be unified if and only if either:
 - One of them is a variable
 - They are identical constants
 - They are structures and
 - * They have the same constructor name
 - * They have the same number of arguments which can be pairwise unified

Example 1: unification

?- 10 = X.

X = 10 ? ;

no

?- 5 + 5 = 10. % +(5,5) = 10.

no

?- node(left,123,X) = node(A,B,B).

A = left, B = 123, X = 123 ? ;

no

Example 2: unification

?- E = 1+2.

E = 1+2;

no

?- p(p(p(p(0)))) = p(p(X)). % Peano arithmetic

X = p(p(0)) ? ;

no

Example 3a: unification

```
% The predicate length1
length1(0, []).
length1(N, [ F|R ] ) :- length1(N2,R) , N = N2+1.
_____

?- length1(X, [a,b,c]).
X = 1+(1+(1+0)) ? ;
no
```

Example 3b: unification

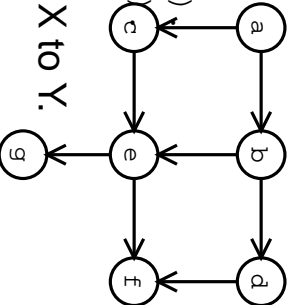
```
% The predicate length
length(0, []).
length(N, [ F|R ] ) :- length1(N2,R) , N is N2+1.
_____

?- length(X, [a,b,c]).
X = 3 ? ;
no
```

Syntax

- **Predicates:** mamma, mMm, no_1_
- **Variables:** w, w12, _12, pi
- **Constants:** stefan, STEFFAN, 'Stefan'
- **Structures:** node(node(X,10,nil),12,nil)
- **Lists:** [], [1,2, [44, 55]], 3], [a, n(1,2,3)]
- **Arithmetic & others:** +, -, *, /, sin, cos, tan, <, >, >=, =<, =:=, =\=, ==, =, is
- **Numbers (integer and real):** 0,..., 9, 502, 123.31, 55.2e-3

Transitive Relations

- **A graph:**
edge(a,b). edge(a,c). edge(b,d).
edge(c,e). edge(d,f). edge(e,g).


```
graph LR; a((a)) --> b((b)); a((a)) --> c((c)); b((b)) --> d((d)); c((c)) --> e((e)); d((d)) --> f((f)); e((e)) --> g((g));
```
- **write a predicate that is true if there is a path from X to Y.**
path(X,X).
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), path(Y,Z).
- **Creates problems for e.g. ?- path(p,q).**
path(p,q) -> path(p,y1),path(y1,q) ->
path(p,y11),path(y11,y1), path(y1,y12),path(y12,q) -> ...

Better solution

```
path(X,X).  
path(X,Z) :-  
    edge(X,Y),  
    path(Y,Z).
```

Debugging new solution

```
?- trace,path(a,f).  
1 1 Call: path(a,f) ?  
2 2 Call: edge(a,_974) ?  
2 2 Exit: edge(a,b) ?  
3 2 Call: path(b,f) ?  
4 3 Call: edge(b,_2096) ?  
4 3 Exit: edge(b,d) ?  
5 3 Call: path(d,f) ?  
6 4 Call: edge(d,_3218) ?  
6 4 Exit: edge(d,f) ?  
7 4 Call: path(f,f) ?  
7 4 Exit: path(f,f) ?  
5 3 Exit: path(d,f) ?  
3 2 Exit: path(b,f) ?  
1 1 Exit: path(a,f) ?
```

yes