IFT 6802
Introduction to Prolog

**Par Jean Vaucher**
**(& Laurent Magnin)**



## Prolog

**A programming language based on the formalism and the concepts of formal logic.**

- PROgrammation LOGique
- Robinson (resolution)
- Kowalski (Logic Programming)
- Colmerauer & Roussel ( PROLOG )
- Warren DEC-10 (Quintus)
- I.C.O.T. PSI Engine
- 5th Generation



## Declarative programming

- **Concise Notation**
- **Programming by specification**
- **Constraints**
- **DB Programming Facts and Rules**



## LOGIC PROGRAMMING BASICS

- **Terms**
  - relationship between objects
    - loves (peter,jane)
- **Variables & functional expressions**
    - loves (X, mother(jane))



## LOGIC PROGRAMMING BASICS (suite)

  - <head> :- <body> .
- **RULES**
  - bird(B) :- flies(B), lays_eggs(B).
- **FACTS (no body)**
  - flies (sparrow).
- **QUERY (no head)**
  - :- loves(X,peter) , girl(X).
    - 1) Prove that there is someone who loves Peter and is a girl
    - 2) Find values of X so that the terms in the query match facts in the database



## Facts

  - parent (john, robert) .
- **relationship (predicate) : parent**
- **objects: John, Robert**
  - John is a parent of Robert
- **Interpretation**
  - A parent of John is Robert. ? ? ?
- **Arity**
  - 2 / parent (john, robert) .
  - 1 / man ( john ) .
  - 0 / hot .

## Program

- Set of Facts
- Database

```
likes ( fred , susan) .
likes ( mary , fred) .
likes ( fred , beer ) .
likes ( susan , fred ) .
likes ( jack , mary) .

man (jack) .
man (fred) .

woman (susan).
woman (mary).

drink (beer) .
```



## Queries

- Is fred a man ?
  - :- man (fred) .
  - => ok
- Is **man(fred)** TRUE ?
- Is the *fact* **man(fred)** in the database ?
- Goal to be proved



## Complex Queries

- Does Fred like Mary ?
  - :- likes (fred , mary ) .
  - **=> no**
- Conjunction of goals
  - :- likes ( fred , beer ) , man ( fred ) .
  - **=> ok**



## Variables

- **An unspecified individual**
- **Denoted by initial Capital letter**
  - Who likes Mary ?
  - Is there an X such likes( X ,mary) can be found in the DB ?
    - :- likes ( X , mary ) .
    - => X = jack



## Variables

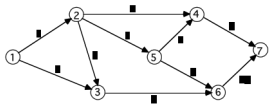
- **Multiple answers**
  - What does Fred like ?
    - :- likes ( fred , T ) .
    - => T = susan ;
    - T = beer
  - What girls does Fred like ?
    - :- likes ( fred , G ) , woman (G) .
    - => G = susan



## PERT Example

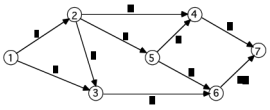
- **Prolog description**
  - **arc ( Id, start-node, end-node, duration, resource)**

```
arc (1,n1,n2,1,r1).
arc (2,n1,n3,4,r2).
arc (3,n2,n3,2,r3).
arc (4,n2,n4,3,r1).
arc (5,n2,n5,1,r2).
arc (6,n3,n6,4,r3).
arc (7,n5,n4,2,r1).
arc (8,n4,n7,3,r2).
arc (9,n5,n6,1,r3).
arc(10,n6,n7,4,r1).
```



## DATABASE PROGRAMMING

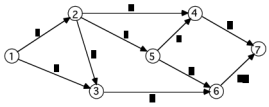
- Does arc4 link nodes 2 and 4 ?
  - **:- arc(4,n2,n4,_,_).**
  - **=>** OK
- What is the duration of activity 6 and what resource does it use ?
  - **:- arc(6,_,_,D,R).**
  - **=>** D=4, R=r3






## DATABASE PROGRAMMING

- Does activity 7 follows directly activity 5.
  - **:- arc(5,_,N,_,_), arc(7,N,_,_,_).**
  - **=>** N=5 **% Yes, they meet at node 5**
- What activities start at node 5 ?
  - **:- arc(A,n5,_,_,_).**
  - **=>** A=7
  - A=9






## PROLOG's Algorithm

- :- $term_1$, $term_2$, . . . , $term_n$ .
  - **equivalent to**
- for_all DB matches for term1 do
  for_all DB matches for term2 do
  . . . .
  for_all DB matches for termn do
  PRINT VALUES OF VARIABLES
  ( and optionally stop );
- BACKTRACKING
  - **When a term** fails**, Prolog uses another match for the previous term and tries again.**



## Example

likes ( mary , fred) .
likes ( fred , susan) .
likes ( fred , beer ) .
likes ( susan , fred ) .
likes ( jack , mary) .

woman (susan).
woman (mary).

:- likes( fred,G), woman(G).

*likes ( mary , fred)*
**likes ( fred , susan)**
**=> woman(susan)**
**woman (susan)**
**==> G=susan**
*woman (mary).*
**likes ( fred , beer )**
**=> woman(beer)**
*woman (susan).*
*woman (mary).*
*likes ( susan , fred ) .*
*likes ( jack , mary) .*



## Rules

- **<head> :- <tail>**
  - Head is true IF tail is true
  - To prove Head , try to prove tail
- **Examples:**
  - friends (A,B) :- likes (A,B) , likes (B,A).
  - person (P) :- woman (P) .
  - person (P) :- man (P) .
  - likes (M, mary) :- man (M) .



## Examples

**likes ( mary , fred) .**
**likes ( fred , susan) .**
**likes ( fred , beer ) .**
**likes ( susan , fred ) .**
**likes ( jack , mary) .**

**woman (susan).**
**woman (mary).**

**man (fred).**
**man (jack).**

**:- friends (susan , F) .**
**=> F = fred**

**:- likes (F , mary) .**
**=> F = jack ;**
**F = fred ;**
**F = jack**

**:- person (X) .**
**=> X = susan ;**
**X = mary ;**
**X = jack;**
**X = fred**

## Operators

- **Infix Notation**
  - 1 + 2
  - '+'(1,2)
- **Unary & Binary operators**
  - op( 500 , yfx , + ) .
- **Useful Operators**
  - arithmetic:
    - + - * /
    - is := evaluable predicates
  - comparison:
    - > < =:=



## Use of Arithmetic Operators

```
:- X is 4 + 5 , Y is ( X - 1 ) * 2 , 2 is 3-1.
   X = 9
   Y = 16
   Yes
```



## Unification

- **Unification is a symmetric operation (X=Y is the same as Y=X), and is not the same as assignment.**
- **Any value can be unified with itself. (This is normally useful only as a test.)**
  - Example: mother(john) = mother(john)
- **A variable can be unified with another variable. The two variable names thereafter reference the same variable.**
  - Example: X = Y, X = 2, write(Y). /* Writes the value 2. */



## Unification

- **Two different values can be unified if there are unifications for the constituent variables which make the values the same.**
  - Example: mother(mary, X) = mother(Y, father(Z)). [Also results in the unifications mary=Y and X=father(Z)].
- **It is legal to unify a variable with an expression containing itself; however, the resultant value cannot be printed, and must otherwise be handled with extreme care.**
  - Example: X = foo(X, Y).



## Instantiation

- **A variable can be unified with any Prolog value; this is called instantiating the variable. A variable is fully instantiated if it is unified with a value that does not itself contain variables.**
  - Example: X = foo(bar, [1, 2, 3]). /* X is fully instantiated. */
  - Example: Pa = husband(Ma). /* Pa is partially instantiated. */



## Unification operator

```
:- f(1) = f(X) , =(Var,123) .
   X = 1
   Var = 123

:- X = 1 + 3 , Y is 1 + 3 .
   X = 1 + 3
   Y = 4
```

```
:- 4 = 1 + 3 .
   no.

:- f (X , g( X , Y)) = f (123 ,
   g(Z , Z )) .
   X = 123
   Y = 123
   Z = 123

:- A + 1 = red + 1 .
   A = red
```

## Complex Unification

*(Zaniolo,,C. 1984 Object-oriented programming in Prolog)*

```
area( rect( H , W ) , A)  :-  A is  H * W .
area( square (Side)  , A)  :-  area( rect(Side,Side), A) .
area( triangle (H, W),  A)  :-  A is  (H * W) / 2 .

 :- X = square(10) , area(X,Z) , write(area=Z) .
   area=100
   ok
```

- **selection of rule**
- **parameter passing (values in / out)**
- **Note use of = to combine terms**



## ==, = & is

- **:- X = 1 + 2 .**
- **X = 1 + 2**

- **:- X == 1 + 2 .**
- **no**

- **:- X is 1 + 2 .**
- **X = 3**

- **:- X is f ( 1 , 2 ) .**
- **ERROR: Arithmetic: `f/2' is not a function**



## Inequality

- **Not unifiable** \=
  - :- joe \= fred.
  - ok
  - :- X \= 123 .
  - no
  - :- 1 + 2 \= 3 .
  - ok
  - :- 1 + 2 \= X .
  - no
- **Not Equal** =:=
  - :- 1 + 2 =:= 3 .
  - no
- **Not Identical** \==
  - :- 1 + 2 \== 3 .
  - ok



## Input / Output

- **write**(+*Term*)
  - **Write *Term* to the current output, using brackets and operators where appropriate.**
- **read**(-*Term*)
  - **Read the next Prolog term from the current input stream and unify it with *Term*.**
- **nl**
  - **Print a new line**
- …



## Input / Output

```
:- write('Input:  ') ,
   read(I),
   I2 is I+1,
   write(I2), nl .

Input:   123 .
124
ok
```



## Assert & Retract

- To update the database during execution

```
:- male(X).
  X = fred ;
  X = toto
:- assert( male(john) ).
:- male(X).
  X = fred ;
  X = toto ;
  X = john
:- retract( male(toto) ).
:- male(X).
  X = fred ;
  X = john
```

## Assert, asserta & assertz

- **assert(+Term)**
  - Assert a fact or clause in the database. Term is asserted as the last fact or clause of the corresponding predicate
- **asserta(+Term)**
  - Equivalent to assert, but Term is asserted as first clause or fact of the predicate.
- **assertz(+Term)**
  - Equivalent to assert.



## Assignment

- **Pascal**
  - Var x;
  - x := 5;
  - writeln (x);
  - x := x+1;
- **Prolog**
  - assert( value_of( x,5 ) ),
  - value_of( x,X ), writeln( X ),
  - retract( value_of( x,OldX )) ,
  - NewX is OldX+1,
  - assert( value_of( x,NewX ) ).

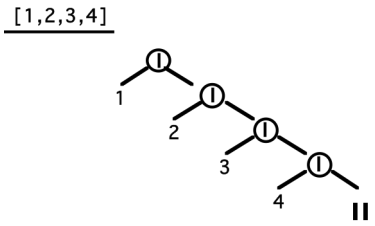


## List Processing

- **Lists:**
  - Dynamic data structures
  - Trees
  - Graphs
- **special symbol for empty list:**
  - nil or []
- **binary nodes: "|" or "."**
  - Left = first
  - Right = rest of list



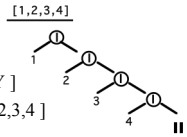
## Example of list



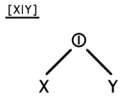



## List Processing

- **Accessing elements**
- **Building lists**




[ 1,2,3,4 ] = [ X|Y ]
=> X =1 , Y = [ 2,3,4 ]




:- [1,2,3 ] = [_,A| B] .
A = 2 , B =[3]



## Procedures for Lists

- **first (E,L)**
  - E is the first element of the list L

first ( X , [X|_] ) .

:- first ( C , [red,white,blue] ) .
C = red

## Procedures for Lists

- **last (E,L)**
  - E is the last element of the list L

```
last( X , [X] ) .
last( X , [_| Rest] ) :- last( X , Rest ) .

:- last ( C , [red,white,blue] ) .
   C = blue
```



## Procedures for Lists

- **member( E,L )**
  - E is a member of the list L

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

:- member( red , [red,white,blue] ).
  ok
```



## Using the Rules

- **Generator**

```
:- member( X , [red,white,blue] ).
   X = red ;
      ➡  member( X , [white,blue])
   X = white ;
      ➡  member( X , [blue])
   X = blue
```

- **Constraints**

```
:- L = [_,_,_] , member( fred,L ), member (3,L), last(zzz,L) .

   L = [fred,3,zzz] ;
   L = [3,fred,zzz]
```



## Logic Black Holes

- **infinite loops after a few good answers**
- **generators of elements of infinite set (i.e. lists)**
- **Example**

```
:- member( 1 ,L ) , L = [X,X] .
   L = [1,1] ;
   L = [1,1] ;
   ..... infinite loop ...
```

  - The first term generates all lists of which "1" is a member:

[1], [1,..], [_,1], [_,1,..], [_,_,1], [_,_,_,1]... etc ...

  - Only two are acceptable, but the generator keeps providing candidates for the second term to reject.



## Control Predicates

- **fail**
  - Always fail. The predicate fail is translated into a single virtual machine instruction.
- **true**
  - Always succeed. The predicate true is translated into a single virtual machine instruction.
- **repeat**
  - Always succeed, provide an infinite number of choice points.
- **!**
  - Cut. Discard choice points of parent frame and frames created after the parent frame.