

LOGIC PROGRAMMING AND PROLOG

PROLOG 1

Summary

- Logic programming
- Knowledge base
- Queries
- Recursive rules
- Program execution
- Operational model

Logic programming

Declarative programming.

- Program = problem description
- Execution = check the truth of an assertion (goal)

R. Kowalski : Algorithm = Logic + Control.

Applications of logic programming

PROLOG is the major logic-based programming language (subset of *First Order Logic*).

Resources (implementation):

<http://www.swi-prolog.org/>

Resources (textbook):

L. Sterling, E. Shapiro, *The Art of Prolog*, 2nd Ed., MIT Press, 1994.

Resources (other book):

<http://www.learnprolognow.org/>

Applications of logic programming

- deductive databases
- expert systems
- knowledge representation for robots!!

Basic intuition

Logic program:

1. definition of the problem through the assertion of **facts** and **rules**;
2. Querying the system which **infers** the answer to the query given known facts and rules (theorem provers).

Aristotelic Syllogism

- *All men are mortal*
- *Socrates is a man*

we can infer: *Socrates is mortal.*

Aristotelic Syllogism in PROLOG

```
mortal(X) :- man(X).  
man(socrates).
```

The inference is started by:

```
?mortal(socrates)
```


Knowledge base

A PROLOG program is composed by a set of **clauses**, i.e. **conditional** and **unconditional assertions**.

unconditional assertion (**fact**):

```
father(daniele,jacopo).
```

```
loves(enzo,X).
```

In PROLOG:

- the names of predicates and constants/individuals start with a lower case letter (e.g. father, jacopo),
- while the variable identifiers start with a capital letter (e.g. X).

Rules

conditional assertion (rule):

$A :- B, C, \dots, D.$

A is true if B, C, ..., D are true,

- A is the **conclusion/head**,
- B, C, ..., D are the **premises/body**
- A, B, C, D are **atoms**

If t_1, \dots, t_n are terms and P is an n-ary predicate $P(t_1, \dots, t_n)$ is an atom.

We start simple (without function symbols), so terms are either constants or variables.

Examples of rules

`grandFather(X,Z) :- father(X,Y), father(Y,Z).`

`grandFather(X,Z) :- father(X,Y), mother(Y,Z).`

`son(X,Y) :- father(Y,X).`

`son(X,Y) :- mother(Y,X).`

`grandFather`, `son` can be seen as **procedures**

Querying the system

goal (query)

? A,B,C,...,D.

? father(daniele,jacopo).

YES.

Knowledge base

```
father(daniele,michela).  
father(daniele,jacopo).  
father(eriberto,daniele).  
father(antonio,eriberto).  
mother(alma,eriberto).  
mother(annamaria,daniele).  
mother(annamaria,marcello).  
mother(annamaria,sandro).
```

```
nice(michela).  
nice(anna).
```

```
fem(michela).
```

Queries

? nice(X) .

YES michela.

to get other answers: ;

YES anna

goal conjunction:

? grandFather(eriberto,X), nice(X) .

? grandFather(X,Z), nice(Z) .

Recursive rules

```
descendant(X,Y):-son(X,Y). % 1
descendant(X,Y):-son(Z,Y),descendant(X,Z). % 2
son(X,Y):-father(Y,X). % 3
son(X,Y):-mother(Y,X). % 4

? descendant(michela,eriberto).
```

Directed Graph

```
/* Directed Graph */
```

```
arc(a,b).
```

```
arc(a,c).
```

```
arc(b,d).
```

```
arc(c,d).
```

```
arc(d,e).
```

```
arc(f,g).
```

```
/* Transitive closure of the arc relation
```

```
connected(Node1,Node2) :- Node1 connected to Node2
```

```
*/
```

```
connected(Node,Node).
```

```
connected(Node1,Node2) :- arc(Node1,NodeInt),  
                           connected(NodeInt,Node2).
```


Multiple roles of the arguments

? descendant(X,daniele).

? descendant(daniele,X).

? descendant(X,Y).

? connected(a,X).

? connected(X,a).

? connected(X,Y).

PROLOG operational model

- abstract interpreter
- search of the solution
- unification

Unification (simplified)

A *substitution* is a function from the set of variables VAR to the set of terms $\text{STERM} = \text{VAR} \cup \text{CONST}$:

$$\sigma : \text{Var} \mapsto \text{STerm}.$$

The substitution σ of a variable X by a term t is denoted by $X = t$ (or X/t).

Given t , $t\sigma$ is defined (without function symbols) as follows:

- if c is a constant symbol, $c\sigma = c$;
- if X is a variable symbol, $X\sigma = \sigma(X)$;
 - $\sigma(X) = X$, if σ does not contain a substitution for X
 - $\sigma(X) = t$, if t is the replacement of X in σ

Unification (simplified)

The substitution that makes two expressions identical is denoted $\theta = \text{unify}(e_1, e_2)$.

Examples

$$\text{unify}(a, a) = \{\}$$

$$\text{unify}(X, a) = \{X/a\}$$

$$\text{unify}(X, Y) = \{X/Y\}$$

$\text{unify}(b, a) = ?$ NO substitution can make a and b identical

Unification (simplified)

Unification is applied to expressions of the form $P(t_1, t_2, \dots, t_n)$.

$unify(P(t_1, t_2, \dots, t_n), P(s_1, s_2, \dots, s_n))$: find a substitution that makes $P(t_1, t_2, \dots, t_n)$ and $P(s_1, s_2, \dots, s_n)$ identical.

Examples

$$unify(P(X), P(a)) = \{X/a\}$$

$$unify(P(X), P(Y)) = \{X/Y\}$$

$$unify(P(a), Q(a)) =? \text{ NO}$$

$$unify(P(X, b), P(a, Y)) = \{X/a, Y/b\}$$

$$unify(P(X, X), P(a, b)) \text{ NO}$$

Abstract interpreter

Input: a goal G and a program P

Output: an instance of G logical consequence of P if it exists,
otherwise NO

begin

$R := G$; % R is called resolvent

finished := false;

prove the goal in the resolvent; (see next slide)

if $R = \{ \}$

then return G

else return NO

end

Prove the goal in the resolvent

```
while not  $R = \{ \}$  and not finished do
begin
  choose a goal  $A$  in the resolvent
  choose a clause  $A' :- B_1, \dots, B_n$  (renaming variables)
    such that  $\theta = \text{unify}(A, A')$ 
  if no more choices
    then finished:=true;
  else begin
    substitute  $A$  with  $B_1, \dots, B_n$  in  $R$ 
    apply  $\theta$  to  $R$  and  $G$ ;
  end
end
```

The search tree

- the root is the initial goal;
- every node has one successor for each clause whose head unifies with a goal in the node. Every successor has a resolvent obtained by the parent node by replacing the chosen goal with the body of the clause, after applying the unifier.

Every node contains a resolvent. If it is empty the node is a *success node*. A node without successors, not a success node, is a *failure node*.

Every success node represents a solution. If the tree cannot be further expanded and it does not have any success node then the goal fails.

The design choices of PROLOG

- the goal to be resolved determines the structure of the search tree;
- the clause determines the order of the successors of a node.

The PROLOG interpreter chooses the goals from left to right and the clauses are chosen wrt the order specified in the program. The resolvent is a stack. The search tree is built depth-first.

Change the rule order

```
grandFather(X,Z) :- father(X,Y), mother(Y,Z).  
grandFather(X,Z) :- father(X,Y), father(Y,Z).
```

```
father(daniele,michela).  
father(daniele,jacopo).  
father(eriberto,daniele).  
father(antonio,eriberto).  
mother(alma,eriberto).  
mother(annamaria,daniele).  
mother(annamaria,marcello).  
mother(annamaria,sandro).
```

Change the order of the conjuncts in the rule body

```
grandFather(X,Z) :- mother(Y,Z), father(X,Y).  
grandFather(X,Z) :- father(X,Y), father(Y,Z).
```

```
father(daniele,michela).  
father(daniele,jacopo).  
father(eriberto,daniele).  
father(antonio,eriberto).  
mother(alma,eriberto).  
mother(annamaria,daniele).  
mother(annamaria,marcello).  
mother(annamaria,sandro).
```

Change the rule order

descendant(X,Y) :- son(X,Y) .

descendant(X,Y) :- son(Z,Y), descendant(X,Z) .

son(X,Y) :- mother(Y,X) .

son(X,Y) :- father(Y,X) .

? descendant(daniele,X) .

Change the order of the conjuncts in the rule body

```
descendant(X,Y):- son(X,Y).
```

```
descendant(X,Y):- descendant(X,Z),son(Z,Y). %1'
```

```
son(X,Y):- father(Y,X).
```

```
son(X,Y):- mother(Y,X).
```

Change the rule order II

```
descendant(X,Y):- descendant(X,Z),son(Z,Y). %1'  
descendant(X,Y):- son(X,Y).  
son(X,Y):- father(Y,X).  
son(X,Y):- mother(Y,X).
```