# Programmazione logica

M. Gabbrielli, S. Martini

*Linguaggi di programmazione:*
*principi e paradigmi*
McGraw-Hill Italia, 2005

Bob Kowalski: "Algorithm = Logic + Control"

- in traditional programming:
  - programmer takes care of both aspects
- in declarative programming:
  - programmer takes care only of Logic:
  - interptreter of the language takes care of Control

# Declarative programming

The task of the programmer is to specify the problem to be solved: we have to ``declare''

what we want to obtain

and not

how we achieve that

# An example

- Problem: arrange three 1s, three 2, … three 9s in sequence so that for all i∈[1,9] there are exactly i numbers between successive occurrences of i

- A solution:

1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7

# An LP program

- A Prolog program which solves previous problem:

```
sequence([_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_]).
% Sequence(X) -> X is a lit of 27 variables

question(S):-
          sequence(S),
          sublist([9,_,_,_,_,_,_,_,_,_,9,_,_,_,_,_,_,_,_,_,9],S),
          sublist([8,_,_,_,_,_,_,_,_,8,_,_,_,_,_,_,_,_,8] ,S),
          sublist([7,_,_,_,_,_,_,_,7,_,_,_,_,_,_,_,7] ,S),
          sublist([6,_,_,_,_,_,_,6,_,_,_,_,_,_,6] ,S),
          sublist([5,_,_,_,_,_,5,_,_,_,_,_,5] ,S),
          sublist([4,_,_,_,_,4,_,_,_,_,4] ,S),
          sublist([3,_,_,_,3,_,_,_,3] ,S),
          sublist([2,_,_,2,_,_,2] ,S),
          sublist([1,_,1,_,1] ,S).
% S is a solution
```

# PROLOG

First and most used real language based on logic programming: PROLOG (Programming in logic).

First PROLOG interpreter developed by Colmerauer and Russel in 1972.

Several Prolog systems (Sicstus, Eclipse …)

# From automatic deduction to logic programming

- The roots of logic programming are in automatic deduction of logical theorems
- Logic programming is based on particular first order logic (FOL) theories whose axioms are expressed in terms of Definite Horn clauses

# A bit of history

- 1930s: Kurt Gödel and Jacques Herbrand study computability based on derivations. Herbrand in his thesis discusses a set of rules for manipulating algebraic equations on terms: sketch of unification

- 1965 Alan Robinson introduces the resolution principle: a complete rule for proving FOL theorems. Automatic deduction starts

- 1974 Robert Kowalski introduces logic programs which use a restricted from of resolution: proving has a ``side effect'' of producing a substitution which give the result of computation

# Logic programming: sintax

**Alphabet**

- **Var** a set of variable symbols : x, y, z
- **Costr** a set of constructor symbols with theri arity, divided in
  **Const** constants, constructors with arity 0: a,b,c,…
  **Fun** functions, constructors with arity > 0: f, g, h, …
- **Pred** a set of predicate symbols with their arity: p,q,r,…
- **Spec** three special symbols:
  - **:-** left implication. Equivalently we use also the symbol ←
  - , conjunction. Equivalently we use also the symbol and
  - . dot. Denotes the end of an instruction

The set **Term** of terms is defined as

Term ::= Var | Const | Fun(Term, …,Term)

ex. x a f(x,a) f(g(x),y) g(f(a,a))

A ground term is a term which do not contain variables

Herbrand universe = set of all ground terms (on a fixed alphabet)

# sintax - 3

The set **Atom** of atoms is defined by

Atom ::= Pred(Term, …, Term)

ex. p(a,x) q(f(g(a),y)) p(a,f(x,a))

Ground atom = atom which does not contain variables

Herbrand base = set of ground atoms (on a fixed alphabet).

The set of **Horn Clauses** is defined by

HClause ::= Atom **:-** Atom , … , Atom. |
**:-** Atom , … , Atom.

Head :- Body (possibly empty)

- Clause having the head are called program clauses (or definite clauses) H :- B, …,B
- Clauses with the empty body are called facts H. (the :- is omitted).
- Clauses without the head ar called goal :- A,…,A (also ?- A, …,A)

Note: a Clause if a formula of the form $\forall$ A \/ A …. ~ A \/ ~ A
A Horn clause is a clause with at most one positive literal (universal quantification is implicit)

- A **Logic Program** is a set of definite clauses
- A **goal** is a cluse with empty head (False)
- A program

```
direct_fligth(paris, damascus).
direct_fligth(firenze,roma).
direct_fligth(firenze, paris).
fligth(X,Y):- direct_fligth(X,Y).
fligth(x,y):- direct_fligth(x,z), fligth(z,y).
```

- Three goals

```
?- direct_fligth(firenze, damascus).   answer no
?- fligth(firenze, damascus).                  answer  yes
?- fligth(firenze, X).  1st answer X = roma ...
```

| logic | PROLOG |
|---|---|
| ← | **:-** (in definite clauses) |
| | **?-** (in goal) |
| /\ (and) | **,** (virgola) |
| | **.** (punto) end of each clause |

# Example

```
∀x,y. flight(x,y) ← flight_direct(x,y)
∀x,y. flight(x,y) ← flight_direct(x,z) and
   flight(z,y)
```

```
flight(venezia,parigi) ← flight_direct(venezia,parigi)
flight(roma,parigi) ← flight_direct(roma,venezia) and
                       flight(venezia,parigi)
```

```
∃y.    ?- flight(roma,y)
∃z.    ?- flight(z,roma)
```

# Another example

```
% sum(x,y,z)  z is the sum of  x and y

sum(0,y,y)
sum(s(x),y,s(z)) :- sum(x,y,z)

?- sum(s(s(0)), s(0), y)      answer y = s(s(s(0)))
?- sum(x, s(0), s(s(0)))      answer x = s(0)
?- sum(x, y, s(s(0)))            answer ? .....
```

# A problem

Find a number AB consisting of two digits such that

- $(AB)^2$ = XYZW (AB power 2 contains four digits) and
- AB = XY + ZW

N.B. Assume that arithmetic operations are already defined

# Solution

```
% number consisting of 2 digits
  two_d(N) :- greater(N,9), greater(100,N)

% square contains  4 digits
  four_d(M) :- greater(N,999), greater(10000,N)
  four_sq(N) :- square(N,M), four_d(M)

%first two digits and last two digits
 first(M,X) :- div(M,100,X)
 last(M,X) :- mod(M,100,X)

  digit_sum(M,Z) :- first(M,X), last(M,Y),sum(X,Y, Z)
  digit_sum_sq(N) :- square(N,M), digitsum(M,N)

  solution(N) ← two_d(N), four_sq(N), digit_sum_sq(N)
```

# Classic semantics of LP

- A logic program is interpreted as a FOL theory which define the meaning of the predicates in the heads: three equivalent way of defining the semantics
  - model-theoretic semantics (via least Herbrand model)
  - operational semantics (via SLD resolution)
  - fixpoint semantics (via $T_P$ operator)

- Computing = providing a proof via SLD refutation. Computation produces a computed answer substitution which contain the result

- We will follow a more pragmatic approach to semantics: procedural interpretation of LP

# Declarative and procedural interpretation of LP

A Horn clause $A \text{ :- } B_1 \ldots B_n$

- Declarative interpretation:
  - for each ground instance of variables,
    if $B_1 \ldots B_n$ are true then A is true
  - semantics = (ground) logical consequences of P
- Procedural interpretation:
  - in order to solve A solve $B_1 \ldots B_n$.
  - A is a procedure call, $B_1 \ldots B_n$ is the procedure body
  - semantics = (ground) atoms which have successful derivations

# Procedural interpretation of LP

A Horn clause $p(t) :- q_1(t_1) \dots q_n(t_n)$

- Procedural interpretation:
  - p is the procedure name
  - t is the parameter
  - the clause is (part of) the procedure definition
  - $q_1(t_1) \dots q_n(t_n)$ is the procedure body
  - $q_i(t_i)$ is a procedure call
  - to solve p(t) solve $B_1 \dots B_n$.

# Parameter passing

- Parameter passing is by (a kind of) call by name:
  - formal parameters substituted by actual ones in the body of the procedure, provided no variable clash arise:
    - A procedure call $p(t_1 \ldots t_n)$
    - A procedure definition which $p(X_1 \ldots X_n) \text{ :- } B_1 \ldots B_n$
    do not share variables with the call
    - The call leads to the evaluation
    of the goal $(B_1 \ldots B_n)\{X_1 / t_1 \ldots X_n / t_n\}$
  - In the more general case unification is needed

# The general case

- A goal $\quad A_1 \ldots A_i, p(t_1 \ldots t_n), A_{i+1} \ldots A_n$

- A procedure definition $\quad p(s_1 \ldots s_n) \text{ :- } B_1, \ldots, B_n$

can be seen as a shorthand for $\quad p(X_1 \ldots X_n) \text{ :- } X_1 = s_1, \ldots, X_n = s_n, B_1, \ldots, B_n$

where = is interpreted as sintactic equality on ground terms

- Then previous definition apply and we obtain the goal

$$(A_1 \ldots A_i, X_1 = s_1, \ldots, X_n = s_n, B_1, \ldots, B_n, A_{i+1} \ldots A_n)\{X_1 / t_1 \ldots X_n / t_n\}$$

# The general case (using mgu)

- A goal $A_1 \ldots\ p(t_1 \ldots t_n)\ \ldots A_n$

- A procedure definition (which do not share variables with the goal ) $p(s_1 \ldots s_n) \text{ :- } B_1 \ldots B_n$

- The call leads to the evalutation of the goal $(A_1 \ldots B_1 \ldots B_n \ldots A_n)\ \theta$

where $\theta = mgu\ ((t_1 \ldots t_n), (s_1 \ldots s_n))$

# Non determinism

- Two sources of non determinism:
  - which procedure call to select in a goal for rewriting

    $p_1 (t_1) \ldots p_i(t_i) \ldots p_n(t_n)$
  - which proceure definition (clause) to use (don'know non determinism):

    $p_1 (t_1)$ :- $B_1$

    $p_1 (t_2)$ :- $B_2$

    $p_1 (t_2)$ :- $B_2$

  - Prolog use
    - leftmost atom in a goal
    - textual order of clauses in the program (top down)
    - Thi implise deep-first search strategy

# Backtracking

- Don’t know non-determinsm is implemented by Backtracking:
  - if a failure arise the computation backtracks to the last choice point (clause selection) and the next clause is tried.
  - if not further choices are available computation fails
  - Backtracking is the main source of inefficiency in program execution

- And for the atom selection ?:
  - no backtracking is needed: any selection rule is ok.