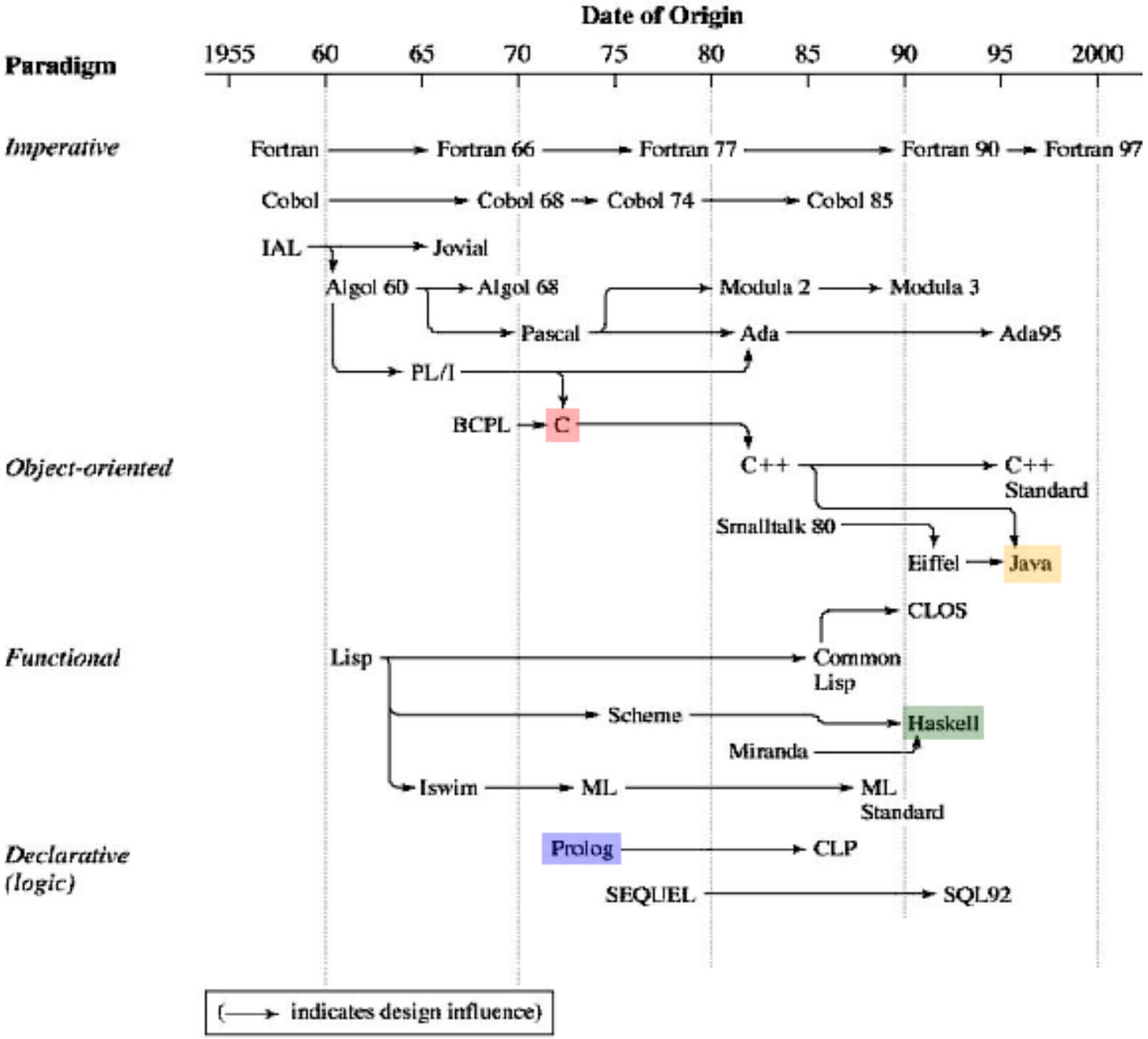
Constraint Programming: Theory and Practice

Topics:
From Prolog to Constraint Logic Programming

Prof. Dr. Slim Abdennadher

History of Programming Languages



Prolog Execution

- Procedural view of clauses: Call with parameter passing is replaced by call with **variable unification**.
- Unification of terms: minimal substitution of variables that make the 2 terms equal.
- No description of the operational aspect: The order to consider clauses and goals is arbitrary (only in theory).
- Use of **backtracking** to explore all alternatives.

LP versus CLP: Constraint Solving instead Unification

- **Prolog Approach**:

```
p(X,Y,Z):- Z is X+Y.

:- p(4,5,Z).
Z=9

:- p(X,5,9).
{IMSTANTIATION ERROR}
```

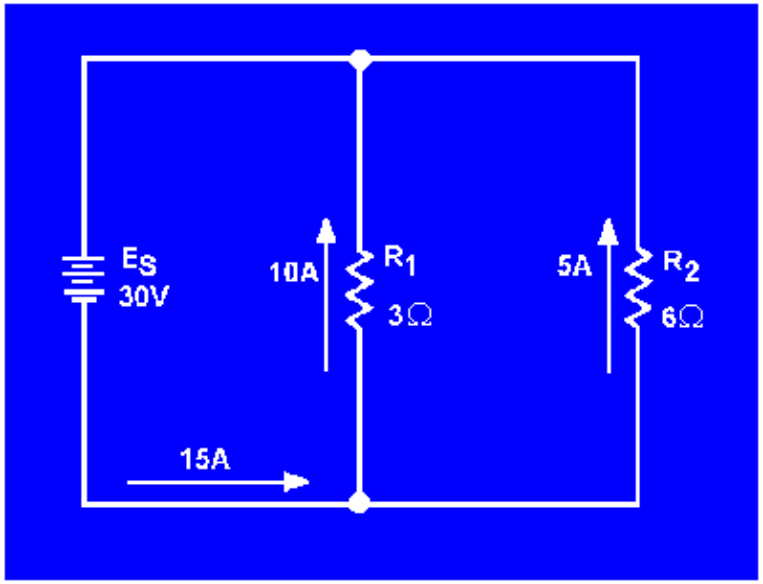
- **CLP approach**:

```
p(X,Y,Z):- Z #= X+Y.

:- p(4,5,Z).
Z=9

:- p(X,5,9).
X=4
```

Model of a Simple Circuit




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

parallel_resistors(V,I,R1,R2) :-
        {V = I1*R1, V = I2*R2, I1+I2 = I}.
```

- Behavior with resistors of 10 and 5 ohms

```
?- parallel_resistors(V,I,R1,R2), {R1 = 10, R2 = 5}.
V = 10/3*I
```

- Behavior with 10V battery where resistors are the same.

```
?- parallel_resistors(10,I,R,R).
```

Model of a Simple Circuit

```
{_B=I-_A},
{10-R*_B=0},
{10-_A*R=0}
```

Model a Triangle?

Define the constraints on the sides of a triangle:

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


triangle(X,Y,Z) :-
    X #> 0, Y #> 0, Z #> 0,
    X+Y #> Z, Y+Z #> X, X+Z #> Y.

| ?- triangle(X,4,5).
X in 2..8 ?
yes
```

Factorial in Prolog

```
fac(0,1).

fac(X,Y) :-
    X > 0,
    X1 is X - 1,
    fac(X1,Y1),
    Y is X*Y1,

| ?- fac(4,L).
L = 24 ?

| ?- fac(L,24).
{IMSTANTIATION ERROR}
```

Factorial using the Finite Domain Solver

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

fac(0,1).

fac(X,Y) :-
    X #> 0,
    Y #= X*Y1,
    X1 #= X - 1,
    fac(X1,Y1),

| ?- fac(4,L).
L = 24 ?

| ?- fac(L,24).
L = 4 ?
yes
```

Fibonacci using the Finite Domain Solver

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

fib(0,1).
fib(1,1).
fib(N,Y) :-
    Y #= X1+X2,
    N #> 1,
    N1 #= N - 1,
    N2 #= N - 2,
    fib(N1,X1),
    fib(N2,X2).

| ?- fib(5,J).
J = 8 ?

| ?- fib(J,89).
J = 10 ?
yes
```

Fibonacci using the Finite Domain Solver

Which arguments give a Fibonacci value in the interval `80 .. 90`?

```
?- 80 #=< B, B #=< 90, fib(A,B).

A = 10,
B = 89 ?
```

Mortgage Example

- P: amount owed
- T:the number of periods in the mortgage
- I: the interest rate of the mortgage
- R: the payment due each period of the mortgage
- B the balance owing at the end

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

mortgage(P, T, _, _, B) :-
  { T = 0, B = P }.
mortgage(P, T, I, R, B) :-
  {
     T >= 1,
     NT = T - 1,
     NP = P + P * I - R
  },
  mortgage(NP, NT, I, R, B).
```

Mortgage Examples

```
| ?- mortgage(3000, 12, 0.05, R, 0).
R = 338.47623006244623

| ?- mortgage(L, 12, 0.05, 1000, 0).
L = 8863.25163644881

| ?- mortgage(10000, M, 0.05, 1000, R), {R =< 0}.
M = 15.0,
R = -789.2817941136727

| ?- mortgage(P, 720, 0.01, B, M).
M=2.0536942403163825*P-1053.6942403164994*B
```

LP versus CLP:

Constrain and Generate instead Generate and test

- **LP approach**

```
solution(X,Y,Z):-
        p(X), p(Y), p(Z),
        test(X,Y,Z).
p(11). p(3). p(7). p(16). p(15). p(14).
test(X,Y,Z):- Y is X+1, Z is Y+1.

:- solution(X,Y,Z).
```

458 steps to get the first solution

- **CLP approach**

```
solution(X,Y,Z):-
        test(X,Y,Z),
        p(X), p(Y), p(Z).
p(11). p(3). p(7). p(16). p(15). p(14).
test(X,Y,Z):- Y #=  X+1, Z #=  Y+1.

:- solution(X,Y,Z).
```

11 steps to get the first solution

Small Sudoku

- **Sudoku** stems from Japan.
- A Sudoku puzzle consists of a square of 81 cells. The square is decomposed in 9 small squares with 9 cells. The small squares are called **blocks**.
- Each block consists of digits from 1 to 9. The cells of a block have to have **distinct** digits.
- The digits in each column or in each row of the big square should occur **only once**.
- Let us play Sudoku with 16 cells.

4	1	3	2
2	3	4	1
3	2	1	4
1	4	2	3

Small Sudoku – CLP Program

C1	C2	C3	C4
C5	C6	C7	C8
C9	C10	C11	C12
C13	C14	C15	C16

```
smallsudoko(L) :-
      L = [C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16],
      domain(L, 1,4),
      C2 #= 1, C4 #= 2,  C5 #= 2, C11 #= 1, C16 #= 3,
      all_different([C1,C2,C3,C4]),
      all_different([C5,C6,C7,C8]),
      all_different([C9,C10,C11,C12]),
      all_different([C13,C14,C15,C16]),
      all_different([C1,C5,C9,C13]),
      all_different([C2,C6,C10,C14]),
      all_different([C3,C7,C11,C15]),
      all_different([C4,C8,C12,C16]).
```

Cryptoarithmetic Puzzle – SEND + MORE = MONEY

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

- Letters stand for distinct digits.
- Numbers do not start with 0.
- The equation $SEND + MORE = MONEY$ should hold.

SEND + MORE = MONEY (Constrain/Generate)

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

send([S,E,N,D,M,O,R,Y]) :-
      domain([S,E,N,D,M,O,R,Y],0,9),
      S #\= 0, M #\= 0,
      all_different([S,E,N,D,M,O,R,Y]),
                    1000*S + 100*E + 10*N + D
      +             1000*M + 100*O + 10*R + E
      #= 10000*M + 1000*O + 100*N + 10*E + Y,
      labeling([],[S,E,N,D,M,O,R,Y]).
```

SEND + MORE = MONEY (Constrain/Generate)

send without Labeling

```
:- send([S,E,N,D,M,O,R,Y]).
M = 1, O = 0, S = 9,
E in 4..7,
N in 5..8,
D in 2..8,
R in 2..8,
Y in 2..8 ?
:- send([9,4,N,D,M,O,R,Y]).
no
```

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

Propagation determines `N = 5` and `R = 8` then failure because `D` has no possible value

SEND + MORE = MONEY (Propagation)

- The equation below initiates **propagation**

  1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E = 10000*M + 1000*O + 100*N + 10*E + Y

- Transform the equation into a simpler (equivalent) one:

  1000*S + 91*E + D + 10*R = 9000*M + 900*O + 90*N + Y

- M should be less than or equal to
  $\frac{1}{9} * max(S) + \frac{91}{9000} * max(E) + \frac{1}{9000} * \max(D) + \frac{1}{900} * max(R) - \frac{1}{10} * min(O) - \frac{1}{100} * min(N) - \frac{1}{9000} * min(Y)$

- Given the current domain this implies that
  $M \leq \frac{9}{9} + \frac{91*9}{9000} + \frac{9}{9000} + \frac{9}{900} - \frac{0}{10} - \frac{0}{100} - \frac{0}{9000} = 1.102$

- Thus the domain of M is updated to `[1..1]`

- S should be greater than $9 * min(M) + \frac{9}{10} * min(O) + \frac{9}{100} * min(N) + \frac{1}{1000} * min(Y) - \frac{91}{1000} * max(E) - \frac{1}{1000} * max(D) - \frac{1}{100} * max(R)$

- Given the current domain, we infer that $S \geq 8.082$

SEND + MORE = MONEY (Constrain/Generate)

Solution:

```
[S,E,N,D,M,O,R,Y] = [9,5,6,7,1,0,8,2]
```

$$\begin{array}{cccccc} & & 9 & 5 & 6 & 7 \\ + & & 1 & 0 & 8 & 5 \\ \hline = & 1 & 0 & 6 & 5 & 2 \end{array}$$

SEND + MORE = MONEY (Generate/Test)

```
send([S,E,N,D,M,O,R,Y]) :-
      domain([S,E,N,D,M,O,R,Y], 0,9),
      labeling([],[S,E,N,D,M,O,R,Y]),
      S #\= 0, M #\= 0,
      all_different([S,E,N,D,M,O,R,Y]),
                    1000*S + 100*E + 10*N + D
      +             1000*M + 100*O + 10*R + E
      #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

95,671,082 choices to find the solution

Methods for Constraint Solving

- CLP(R)
  - Gauss-Jordan elimination,
  - simplex
- CLP(FD)
  - arc, node and path consistency methods
  - constraint propagation (forward checking, look-ahead)
  - branch-and-bound
- CLP(B)
  - Operations on Binary Decisions Diagrams (BDDâs)