Constraint Programming: Examples of Programs

Eric MONFROY

IRIN, Université de Nantes

Objectives

- to illustrate the use of ECL^{i}PS^e with examples
- to show the slite difference between modellings (without language) and programs
- to show examples of ECL^{i}PS^e
- to show a kind of methodology for constraint programming

Examples of programs

DONALD + GERALD = ROBERT (1)

- cryparithmetic problem over integers
- replace each letter by a different digit such that

	D	O	N	A	L	D
+	G	E	R	A	L	D
=	R	O	B	E	R	T

is a correct sum

DONALD + GERALD = ROBERT (2)

modelling :

- variables : $D, O, N, A, L, G, E, R, B, T$
- integer domains :
  $[1..9]$ for D and G
  $[0..9]$ for O, N, A, L, E, R, B, T
- constraint :

$$
\begin{aligned}
& 100000.D + 10000.O + 1000.N + 100.A + 10.L + D \\
+ & 100000.G + 10000.E + 1000.R + 100.A + 10.L + D \\
= & 100000.R + 10000.O + 1000.B + 100.E + 10.R + T
\end{aligned}
$$

DONALD + GERALD (in GNU Prolog)

```
donald(LD):-
    LD=[D,O,N,A,L,G,E,R,B,T],
    fd_all_different(LD),
    fd_domain(LD,0,9),
    fd_domain([D,G],1,9),

        100000*D+10000*O+1000*N+100*A+10*L+D +
        100000*G+10000*E+1000*R+100*A+10*L+D
    #= 100000*R+10000*O+1000*B+100*E+10*R+T,
    labeling(LD).
```

DONALD + GERALD (in ECL^{i}PS^e)

```
donald(LD):-
        LD=[D,O,N,A,L,G,E,R,B,T],

        %domains
        LD::[0..9],
        [D,G]::[1..9],

        % constraints
        alldifferent(LD),
           100000*D+10000*O+1000*N+100*A+10*L+D
        +  100000*G+10000*E+1000*R+100*A+10*L+D
        #= 100000*R+10000*O+1000*B+100*E+10*R+T,

        % enumeration
        labeling(LD).
```

DONALD + GERALD = ROBERT (solution)

```
[eclipse 3]: donald(L).

L = [5, 2, 6, 4, 8, 1, 9, 7, 3, 0]
More (0.17s cpu) ? ;

No (0.20s cpu)
```

n-Queens

Place n queens on a $n \times n$ board so that they do not attack each other

Modelling :

- **Variables** : $c_1, \ldots, c_n$

  one per column : the value of c_i represents the line where the queen is in the column

- **Domains** : [1..n]

- **Constraints** : for $i \in [1..n-1)$ and $j \in i+1..n]$
  - not two queens on the same line : $x_i \neq x_j$
  - not 2 queens on the same SW-NE diagonal : $x_i \neq x_j + j - i$
  - not 2 queens on the same NW-SE diagonal : $x_i \neq x_j + i - j$

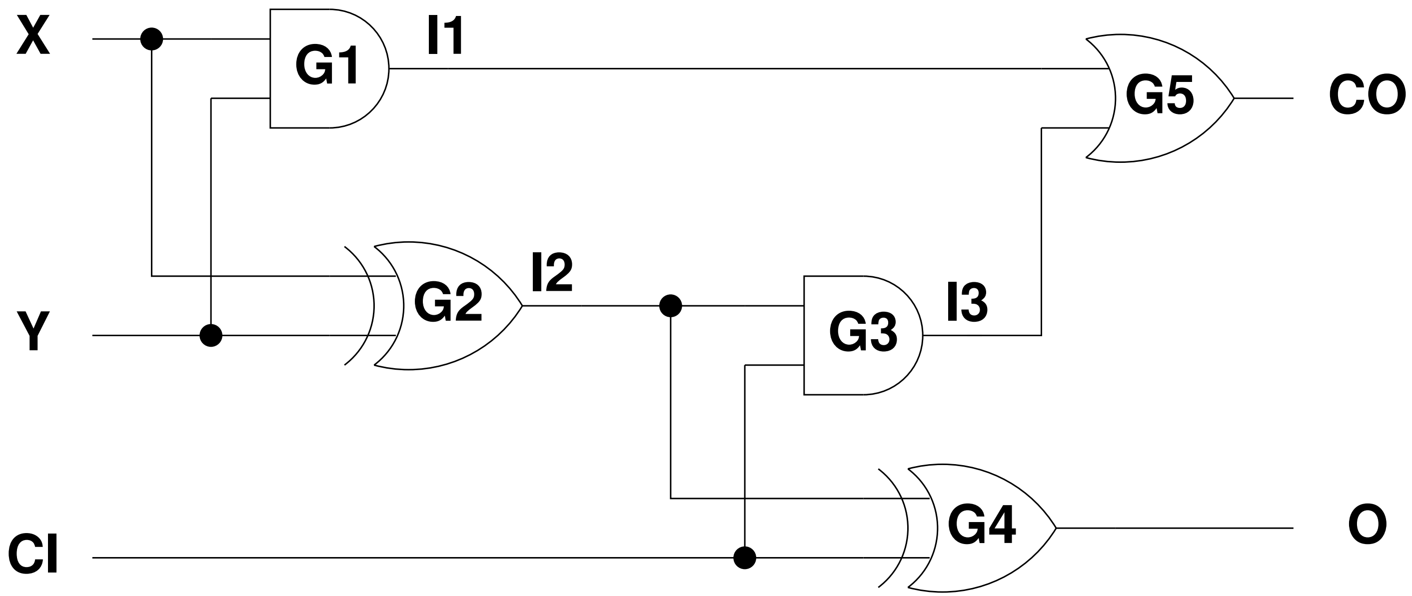
8-queens (ECL^{i}PS^e)

```
queens(L):-
          length(L,8),
          L::[1..8],                     % domains
          safe(L),                       % constraints
          labeling(L).                   % enumeration

safe([]).                                % for i = 1 to n-1 (7)
safe([X|L]):-
          noattack(L,X,1),
          safe(L).

noattack([],_,_).                        % for j = i+1 to n (8)
noattack([Y|L],X,I):-
          diff(X,Y,I),
          I1 is I+1,
          noattack(L,X,I1).

diff(X,Y,I):-                            % constraints
          X#\=Y,                         % not on the same line
          X#\=Y+I,                       % SW-NE diagonal (j-i=I1)
          X+I#\=Y.                       % NW-SE diagonal (j-i=I1)
```

8-queens (solution)

```
[eclipse 6]: queens(L).

    L = [1, 5, 8, 6, 3, 7, 2, 4]
    More (0.00s cpu) ? ;

    L = [1, 6, 8, 3, 7, 4, 2, 5]
    More (0.00s cpu) ? ;

    L = [1, 7, 4, 6, 8, 2, 5, 3]
    More (0.00s cpu) ? ;

    L = [1, 7, 5, 8, 2, 4, 6, 3]
    More (0.00s cpu) ? ;

    ...
```

Full-adder (1)



Full-adder : modelling (2)

Modelling :

- **Variables** : inputs/outputs of gate
- **Domains** : [0,1]
- **Boolean constraints** :
  - $(I1 \iff X \wedge Y), \ (I2 \iff X \oplus Y), \ (I3 \iff I2 \wedge CI)$ $(O \iff I2 \oplus CI), \ (CO \iff I1 \vee I3)$
  - or

    $and(X,Y,I1), \ xor(X,Y,I2), \ and(I1,CI,I3)$ $xor(I2,CI,O), \ or(I1,I3,CO)$

Full-adder : program (3)

```
% load solvers
:-lib(fd).
:-lib(chr).
:-chr("chr/bool").

  fulladder(L):-
        L=[X,Y,CI,I1,I2,I3,CO,O],
        L::[0,1],

        % constraints
        and(X,Y,I1), xor(X,Y,I2), and(I2,CI,I3),
        xor(I2,CI,O), or(I1,I3,CO)

        % search
        labeling(L).
```

Full-adder : solution (3)

```
[eclipse 25]: fulladder(L).
L = [0, 0, 0, 0, 0, 0, 0, 0]
More (0.00s cpu) ? ;
L = [0, 0, 1, 0, 0, 0, 0, 1]
More (0.00s cpu) ? ;
L = [0, 1, 0, 0, 1, 0, 0, 1]
More (0.00s cpu) ? ;
L = [0, 1, 1, 0, 1, 1, 1, 0]
More (0.00s cpu) ? ;
L = [1, 0, 0, 0, 1, 0, 0, 1]
More (0.00s cpu) ? ;
L = [1, 0, 1, 0, 1, 1, 1, 0]
More (0.00s cpu) ? ;
L = [1, 1, 0, 1, 0, 0, 1, 0]
More (0.00s cpu) ? ;
L = [1, 1, 1, 1, 0, 0, 1, 1]
Yes (0.00s cpu)
```

Zebra puzzle : problem (1/2)

1 A small street is composed of 5 colored houses.

2 Five men of different nationalities live in these five houses.

3 Each man has a different profession.

4 Each man likes a different drink.

5 Each man has a different pet animal.

Zebra puzzle : problem (2/2)

6. The Englishman lives in the red house.
7. The Spaniard has a dog.
8. The Japanese is a painter.
9. The Italian drinks tea.
10. The Norwegian lives in the first house on the left.
11. The owner of the green house drinks coffee.
12. The green house is on the right of the white house.
13. The sculptor breeds snails.
14. The diplomat lives in the yellow house.
15. They drink milk in the middle house.
16. The Norwegian lives next door to the blue house.
17. The violonist drinks fruit juice.
18. The fox is in the house next to the doctor’s.
19. The horse is in the house next to the diplomat’s.

Who has the zebra and who drinks water ?

Zebra puzzle : modelling (1/3)

Variables : 25 (5x5)

- men : englishman, spaniard, japanese, italian, norwegian
- profession : painter, sculptor, diplomat, violonist, doctor
- drink : tea, coffee, milk, juice, **water**
- pet animal : dog, snail, fox, horse, **zebra**
- colour : red, green, white, yellow, blue

Domains : [1..5]   (5 houses)

Zebra puzzle : modelling (2/3)

1. A small street is composed of 5 colored houses
   `all_different`(red, green, white, yellow, blue)
2. Five men of different nationalities live in these five houses.
   `all_different`(englishman, spaniard, japanese, italian, norwegian)
3. Each man has a different profession.
   `all_different`(painter, sculptor, diplomat, violonist, doctor)
4. Each man likes a different drink.
   `all_different`(tea, coffee, milk, juice, water)
5. Each man has a different pet animal.
   `all_different`(dog, snail, fox, horse, zebra)

Domains : [1..5]

Zebra puzzle : modelling (3/3)

6	The Englishman lives in the red house.	englishman=red
7	The Spaniard has a dog.	spaniard=dog
8	The Japanese is a painter.	japanese=painter
9	The Italian drinks tea.	italian=tea
10	The Norwegian lives in the first house on the left.	norwegian=1
11	The owner of the green house drinks coffee.	green=coffee
12	The green house is on the right of the white house.	green=white+1
13	The sculptor breeds snails.	sculptor=snail
14	The diplomat lives in the yellow house.	diplomat=yellow
15	They drink milk in the middle house.	milk=3
16	The Norwegian lives next door to the blue house.	$\|$norwegian - blue$\|$ = 1
17	The violonist drinks fruit juice.	violonist = juice
18	The fox is in the house next to the doctor's.	$\|$fox - doctor$\|$=1
19	The horse is in the house next to the diplomat's.	$\|$horse - diplomat$\|$ =1

Zebra puzzle : program (domains)

```
zebra(Nat,Drink,Colour,Prof,Pet):-
        Nat=[Englishman,Spaniard,Japanese,Italian,Norwegian],
        Drink=[Tea, Coffee, Milk, Juice, Water],
        Colour=[Red, Green, White, Yellow, Blue],
        Prof=[Painter,Sculptor,Diplomat,Violonist,Doctor],
        Pet=[Dog, Snail, Fox, Horse, Zebra],
        flatten([Nat,Drink,Colour,Prof,Pet], Var),

        % domains
        Var :: [1..5],

        ...
```

Zebra puzzle : program (constraints)

```
% different values constraint
alldifferent(Nat),
alldifferent(Drink),
alldifferent(Colour),
alldifferent(Prof),
alldifferent(Pet),

% constraints
Englishman #= Red,
Spaniard #= Dog,
Japanese #= Painter,
Italian #= Tea,
Norwegian #= 1,
Green #= Coffee,
Green #= White+1,
Sculptor #= Snail,
Diplomat #= Yellow,
Milk #= 3,
Violonist #= Juice,
```

Zebra puzzle : program (absolute value)

first possibility for the absolute values in constraints

use of reified constraints

```
% to handle constraints with abolute values
(Norwegian - Blue#>=0)  #<=> (Norwegian - Blue #= 1),
(Norwegian - Blue#<0)   #<=> (Norwegian - Blue #= -1),

(Fox-Doctor #>= 0) #<=> (Fox-Doctor #=1),
(Fox-Doctor #< 0)  #<=> (Fox-Doctor #= -1),

(Horse - Diplomat #>=0) #<=> (Horse - Diplomat #=1),
(Horse - Diplomat #<0)  #<=> (Horse - Diplomat #= -1),
```

Zebra puzzle : program (absolute value)

second possibility for the absolute values in constraints

use of carry with domain [-value,value]

```
    % to handle constraints with abolute values
    Dist1 :: [-1,1],
    Dist1 #= Norwegian - Blue,

    Dist2 :: [-1,1],
    Dist2 #= Fox-Doctor,

    Dist3 :: [-1,1],
    Dist3 #= Horse - Diplomat,
```

Zebra puzzle : program (absolute value)

third possibility for the absolute values in constraints

use of Prolog disjunction

→ **bactracking between DIFFERENT CSPs**

```
% to handle constraints with abolute values
(1 #= Norwegian - Blue ; -1 #= Norwegian - Blue),
(1 #= Fox-Doctor ; -1 #= Fox-Doctor),
(1 #= Horse - Diplomat ; -1 #= Horse - Diplomat),
```

Zebra puzzle : program (search)

don't forget the labeling ! ! !

```
    % search
    labeling(Var).
```

Zebra puzzle : solution

```
[eclipse 10]: zebra(N,D,C,P,Pe).

    N = [3, 4, 5, 2, 1]
    D = [2, 5, 3, 4, 1]
    C = [3, 5, 4, 1, 2]
    P = [5, 3, 1, 4, 2]
    Pe = [4, 3, 1, 2, 5]
    More (0.00s cpu) ? ;

    No (0.00s cpu)
```

Magic series : problem

- a magic serie is a sequence $X_0, \ldots, X_{n-1}$ such that each X_i corresponds to the number of occurrences of the number i in the serie.
- We have :

$$X_i = \sum_{i=0}^{n-1} (X_j \doteq i)$$

where $X \doteq Y$ is 1 if $X = Y$ and 0 if $X \neq Y$.

- example for $n = 5$: $[2, 1, 2, 0, 0]$

Magic series : program

use of a reified constraint :

```
magic(N,L):-
        length(L,N),
        L::[0..N],
        constraints(L,L,0),
        labeling(L).

constraints([],_,_).               % for each i
constraints([X|Xs],L,I):-
        sum(L,I,X),
        I1 is I+1,
        constraints(Xs,L,I1).

sum([],_,0).                       % Xi (S) is the number of i
sum([X|Xs],I,S):-
        sum(Xs,I,S1),
        X#=I #<=> B,               % if Xj=I count 1, else 0
        S #= B+S1.                 % sum the number of Xj=I
```

Magic series : solution (1)

```
[eclipse 17]: magic(4,L).

L = [1, 2, 1, 0]
More (0.00s cpu) ? ;

L = [2, 0, 2, 0]
More (0.00s cpu) ? ;

No (0.00s cpu)
```

Magic series : solution (2)

```
[eclipse 18]: magic(5,L).

L = [2, 1, 2, 0, 0]
More (0.00s cpu) ? ;

No (0.00s cpu)


[eclipse 20]: magic(19,L).

L = [15, 2, 1, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 1, 0, 0, 0]
More (7.10s cpu) ? ;

No (7.96s cpu)
```

Magic squares

- easy to program
- several ways to improve :
  - redundant constraints : first compute the sum
  - symmetries : put constraints (ordering) on corners
  - labeling, variable selection : start labeling by the center (most constraint), then diagonals
  - labeling, value choice : biggest value, or mid value