

# Constraint Logic Programming (CLP) in SWI-PROLOG

Original by Albert Oliveras and Robert Nieuwenhuis

GRAU-LI — Fall semester 2017/2018

# Contents

<b>1</b>	<b>Introducció</b>	<b>3</b>
1.1	Exemple 1: problema de les 8 reines ( $\equiv$ tauler d'escacs) . . . . .	3
1.2	Exemple 2: sudokus . . . . .	4
<b>2</b>	<b>Definició de variables i dels dominis on prenen valors</b>	<b>7</b>
2.1	Variables FD en SWIPROLOG . . . . .	7
<b>3</b>	<b>Declaració de les restriccions</b>	<b>7</b>
<b>4</b>	<b>Generació de candidats a solucions (etiquetatge)</b>	<b>9</b>

# 1 Introducció

Definir un problema de CLP consisteix en:

- Definir quines variables es faran servir i el seu significat
- Determinar el domini de les variables
- Establir les restriccions entre les variables
- Els programes en CLP són bàsicament la definició del problema. En GNU Prolog, els programes tenen la següent estructura:
  - Definició dels dominis de les variables
  - Declaració de les restriccions entre les variables
  - Generació de solucions
- Per tal de carregar la llibreria de dominis finits corresponent, cal incloure al principi del fitxer PROLOG:

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

## 1.1 Exemple 1: problema de les 8 reines ( $\equiv$ tauler d'escacs)

Donat un tauler quadrat amb 8x8 caselles, disposar 8 reines sense que es matin entre elles.

- A cada columna no pot haver-hi dues reines, i per tant podem assignar files a columnes. La variable  $X_i$  és la fila corresponent a la columna  $i$ .
- Cadascuna d'aquestes variables pot prendre valors entre 1 i 8 (les files possibles)
- Les restriccions sobre aquestes variables són:
  - $X_i \neq X_j$  si  $i < j$  (no n'hi ha 2 a la mateixa fila)
  - $X_i \neq X_j - (j - i)$ , (no n'hi ha 2 a la mateixa diagonal)  $X_i \neq X_j + (j - i)$  si  $i < j$
  - La línia de  $(a, b)$  a  $(a', b')$  és paral·lela a  $(1, 1) \iff (a' - a, b' - b)$  és múltiple de  $(1, 1) \iff a' - a = s = b' - b \iff b = b' - (a' - a)$
  - La línia de  $(a, b)$  a  $(a', b')$  és paral·lela a  $(1, -1) \iff (a' - a, b' - b)$  és múltiple de  $(1, -1) \iff a' - a = s = b - b' \iff b = b' + (a' - a)$

(la restricció que no hi ha dues reines en una mateixa columna es satisfà per la formalització)

- Programa Prolog que resol això:

```
reines :-
    L = [X1, X2, X3, X4, X5, X6, X7, X8],
%    L = [_ , _ , _ , _ , _ , _ , _ , _],

    L ins 1..8,      %      <---- defineix dominis de variables
    segur(L),        %      <---- estableix les restriccions
    label(L),         %      <---- genera solucions
    write(L),         %      <---- escriu solucions
    nl.

segur([]).
segur([X|L]) :-
    no_ataca(X, L, 1),
    segur(L).

no_ataca(_, [], _).
no_ataca(X, [Y|L], I) :-
    X #\= Y,          %      <--- X i Y s n diferents
    X #\= Y + I,      %      <--- X i Y + I s n diferents
    X #\= Y - I,      %      <--- X i Y - I s n diferents
    J is I + 1,
    no_ataca(X, L, J).
```

## 1.2 Exemple 2: sudokus

Donat un tauler quadrat 9x9, es tracta d'omplir-lo amb xifres de 1 a 9 de manera que no hi hagi xifres repetides en cap fila, en cap columna ni en cap quadrat 3x3. A més, algunes de les caselles ja tenen assignada la xifra que els pertoca.

- Les variables són  $X_{ij}$ , que representen la xifra a la casella de la fila  $i$ , columna  $j$ .
- Les variables  $X_{ij}$  poden prendre valors entre 1 i 9 (les xifres)
- Les restriccions són que les  $X_{ij}$  pertinents siguin diferents a cada:
  - fila ( $f \rightarrow X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{16}, X_{17}, X_{18}, X_{19}$ )

- columna ( $c \rightarrow X_{11}, X_{21}, X_{31}, X_{41}, X_{51}, X_{61}, X_{71}, X_{81}, X_{91}$ )
- quadrat 3x3 ( $(1,1) \rightarrow X_{11}, X_{21}, X_{31}, X_{12}, X_{22}, X_{32}, X_{13}, X_{23}, X_{33}$ )
- + concordança amb les caselles ja plenes
- Programa Prolog que resol això:

```
exemple :-
sudoku([5,3,_,_,7,_,_,_,_,
        6,_,_,1,9,5,_,_,_,
        _,9,8,_,_,_,_,6,_,
        8,_,_,_,6,_,_,_,3,
        4,_,_,8,_,3,_,_,1,
        7,_,_,_,2,_,_,_,6,
        _,6,_,_,_,_,2,8,_,
        _,_,_,4,1,9,_,_,5,
        _,_,_,_,8,_,_,7,9]).

sudoku(L) :-
    L = [X11, X12, X13, X14, X15, X16, X17, X18, X19,
          X21, X22, X23, X24, X25, X26, X27, X28, X29,
          X31, X32, X33, X34, X35, X36, X37, X38, X39,
          X41, X42, X43, X44, X45, X46, X47, X48, X49,
          X51, X52, X53, X54, X55, X56, X57, X58, X59,
          X61, X62, X63, X64, X65, X66, X67, X68, X69,
          X71, X72, X73, X74, X75, X76, X77, X78, X79,
          X81, X82, X83, X84, X85, X86, X87, X88, X89,
          X91, X92, X93, X94, X95, X96, X97, X98, X99],

    % Es defineixen els dominis de les variables.
    L ins 1..9,

    % Es donen les restriccions.
    %
    % all_different(L) for a a que les variables de la llista L
    % siguin totes diferents entre si.
    % NOTA: Fer això en un bucle!
    % Files.
    all_different([X11,X12,X13,X14,X15,X16,X17,X18,X19]),
    all_different([X21,X22,X23,X24,X25,X26,X27,X28,X29]),
    all_different([X31,X32,X33,X34,X35,X36,X37,X38,X39]),
    all_different([X41,X42,X43,X44,X45,X46,X47,X48,X49]),
```

```

all_different([X51,X52,X53,X54,X55,X56,X57,X58,X59]),
all_different([X61,X62,X63,X64,X65,X66,X67,X68,X69]),
all_different([X71,X72,X73,X74,X75,X76,X77,X78,X79]),
all_different([X81,X82,X83,X84,X85,X86,X87,X88,X89]),
all_different([X91,X92,X93,X94,X95,X96,X97,X98,X99]),

% Columnes.
all_different([X11,X21,X31,X41,X51,X61,X71,X81,X91]),
all_different([X12,X22,X32,X42,X52,X62,X72,X82,X92]),
all_different([X13,X23,X33,X43,X53,X63,X73,X83,X93]),
all_different([X14,X24,X34,X44,X54,X64,X74,X84,X94]),
all_different([X15,X25,X35,X45,X55,X65,X75,X85,X95]),
all_different([X16,X26,X36,X46,X56,X66,X76,X86,X96]),
all_different([X17,X27,X37,X47,X57,X67,X77,X87,X97]),
all_different([X18,X28,X38,X48,X58,X68,X78,X88,X98]),
all_different([X19,X29,X39,X49,X59,X69,X79,X89,X99]),

% Quadrats 3x3.
all_different([X11,X21,X31,X12,X22,X32,X13,X23,X33]),
all_different([X14,X24,X34,X15,X25,X35,X16,X26,X36]),
all_different([X17,X27,X37,X18,X28,X38,X19,X29,X39]),
all_different([X41,X51,X61,X42,X52,X62,X43,X53,X63]),
all_different([X44,X54,X64,X45,X55,X65,X46,X56,X66]),
all_different([X47,X57,X67,X48,X58,X68,X49,X59,X69]),
all_different([X71,X81,X91,X72,X82,X92,X73,X83,X93]),
all_different([X74,X84,X94,X75,X85,X95,X76,X86,X96]),
all_different([X77,X87,X97,X78,X88,X98,X79,X89,X99]),

% Es generen els candidats a solucions.
label(L),

% S'Escriu la soluci .
pinta(L).

pinta(L):-pinta_aux(L,9).
pinta_aux([],_).
pinta_aux(L,0):-L\=[],nl,pinta_aux(L,9).
pinta_aux([X|L],N):-
    N>0,write(X),write(' '),
    N1 is N-1,pinta_aux(L,N1).

```

- Estructura dels programes Prolog en Constraint Programming
  - Es defineixen les variables i els dominis on prenen valors
  - Es donen les restriccions sobre aquestes variables
  - Es generen candidats a solucions

## 2 Definició de variables i dels dominis on prenen valors

### 2.1 Variables FD en SWIPROLOG

- SWIPROLOG té un tipus especial de variables, les variables FD (Finite Domain), que només poden prendre valors en els dominis respectius.
- Per defecte, el domini d'una variable FD són tots els enters. De tota manera, es recomana declarar el domini de cada variable FD. Tenim dos predicats per a fer-ho (in, ins):

```

X in -2..4           %---> X pertany a [-2,4]
X in -2..4 \ / 5..8  %---> X pertany a [-2,4] U [5,8]
[X,Y] ins -2..4 \ / 5..8 %---> ambdues variables reben el
                        %      mateix domini [-2,4] U [5,8]

```

- Les variables FD són compatibles amb els enters i amb variables Prolog normals (per aquest motiu no cal declarar-les de manera especial).
- Durant l'execució del programa, el domini d'una variable FD es va reduint pas a pas gràcies a les restriccions.

## 3 Declaració de les restriccions

Les restriccions tenen com a component bàsic les expressions aritmètiques.

- Expressions aritmètiques:  
Una expressió aritmètica FD és un terme Prolog construït a partir d'enters, variables i functors que representen funcions aritmètiques. Les expressions compostes són del tipus:

```

- E
E1 + E2
E1 - E2
E1 * E2
E1 ** E2      %E1 elevat a E2
min(E1, E2)
max(E1, E2)
dist(E1, E2)  %|E1 - E2|
E1 // E2      %divisi entera de E1 entre E2
E1 rem E2     %residu de E1 entre E2

```

A partir de les expressions aritmètiques, es poden construir restriccions aritmètiques.

- Restriccions aritmètiques

```

E1 #= E2      %for a a que E1 sigui igual a E2
E1 #\= E2     %for a a que E1 sigui diferent a E2
E1 #< E2      % ...
E1 #> E2
E1 #=< E2
E1 #>= E2

```

Les restriccions es poden compondre amb operadors booleans per formar restriccions més complexes.

- Restriccions booleanes

```

0          %fals
1          %cert
#\ E       %no E
E1 #/\ E2  %E1 and E2
E1 #\ / E2 %E1 or E2
E1 #==> E2 %E1 implica E2
E1 #<=> E2 %E1 equivalent a E2
E1 #\<=> E2 %E1 diferent de E2

```

De vegades convé forçar a que el nombre de restriccions que es fan certes sigui un cert nombre.

- Altres restriccions:

```

all_different(List)  %for a a que totes les variables de List
                    preguin valors diferents.

```



```

element_var(I, L, X)    %for a a que X sigui igual al I- sim ele
                        (comen ant per 1) de la llista L.

```

## 4 Generació de candidats a solucions (etiquetatge)

`labeling(Opts, Vars)` assigna un valor a cada variable `X` de la llista `Vars` d'acord amb la llista d'opcions `Opts`. El domini de tota variable ha de ser finit. Les opcions permeten controlar el procés de cerca. Diferents categories existeixen per determinar:

- Quina és la variable a instanciar: (`leftmost`, `ff`, `ffc`, `min`, `max`)<sup>1</sup>
- Quin valor s'escull per instanciar (`up`, `down`)
- L'estratègia de branching: (`step`, `enum`, `bisect`)

Com a molt una opció de cada categoria es pot especificar, i cap opció pot aparèixer repetida. L'ordre de les solucions es pot influenciar amb:

```

min(Expr)      max(Expr)

```

Això genera solucions en ordre ascendent/descendent amb respecte a l'avaluació de l'expressió aritmètica `Expr`. El `labeling` ha d'instanciar totes les variables que apareixen en `Expr`. Si s'especifiquen diverses opcions d'aquest tipus, s'interpreten d'esquerra a dreta, e.g:

```

?- [X,Y] ins 10..20, labeling([max(X),min(Y)], [X,Y]).

```

Genera solucions en ordre descendent de `X` i, per cada `X`, les solucions es generen en ordre ascendent de `Y`.

---

<sup>1</sup>veure <http://www.swi-prolog.org/man/clpfd.html> per una descripció detallada