**Master ILI et SIA – Université d'Artois** **2009/2010**
**Programmation par contraintes**

**TD : programmation logique par contraintes – domaines finis**
**(Correction)**

**Exercice 1 : (Zebra problem)**
Cinq personnes de nationalités différentes habitent 5 maisons différentes (qui sont sur le même côté de la rue) de couleur différente. Chaque personne a une profession différente, un animal préféré différent, une boisson préférée différente.

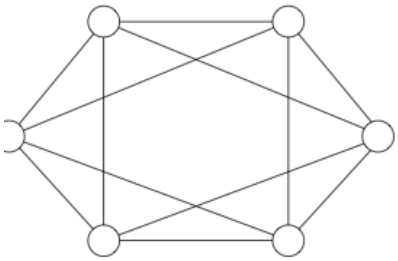
- L'anglais habite la maison rouge.
- L'espagnol a un chien.
- Le japonais est peintre.
- L'italien boit du thé.
- Le norvégien habite la première maison.
- L'habitant de la maison verte boit du café.
- La maison verte vient après la maison blanche.
- Le sculpteur élève des escargots.
- Le diplomate habite la version jaune.
- Dans la troisième maison on boit du lait.
- La maison du norvégien est à côté de la maison bleue.
- Le violoniste boit du jus d'orange.
- Il y un renard dans la maison à côté de celle du docteur.
- Il y a un cheval dans la maison à côté de celle du diplomate.

Écrire un prédicat zebra(X) qui résout ce puzzle, et qui unifie X avec le numéro de la maison dans laquelle est le zèbre. On attend de vous d'écrire un programme en Sictus-Prolog, on n'attend pas que vous trouvez le numéro de la maison du zèbre vous-même !
Indication : Les maisons ont les numéros 1 à 5. Vous utiliserez plusieurs variables à domaine fini; chacune des variables dénote un numéro de maison.

```
:-use_module(library(clpfd)).
:-use_module(library(lists), [append/3]).
maison(Zebre,Eau,Type):-
        Nationalite = [Anglais,Italien,Japonais,Espagnol,Norvegien],
        Couleur = [Rouge,Vert,Bleu,Jaune,Blanc],
        Boisson = [The,Cafe,Lait,JusDeFruit,Eau],
        Animaux = [Renard,Chien,Serpent,Cheval,Zebre],
        Profession = [Diplomate,Peintre,Violoniste,Sculpteur,Docteur],
        append(Nationalite, Couleur, X1),
        append(X1, Boisson, X2),
        append(X2,Animaux,X3),
        append(X3, Profession, Vars),
        domain(Nationalite,1,5),
        domain(Couleur,1,5),
        domain(Boisson,1,5),
        domain(Animaux,1,5),
        domain(Profession,1,5),
        all_different(Nationalite),
        all_different(Couleur),
        all_different(Boisson),
        all_different(Animaux),
        all_different(Profession),
        Anglais#=Rouge,
        Espagnol#=Chien,
        Japonais#=Peintre,
        Italien#=The,
        Norvegien#=1,
```

```
        Vert#=Cafe,
        Vert#=Blanc+1,
        Sculpteur#=Serpent,
        Diplomate#=Jaune,
        Lait#=3,
        voisin(Norvegien,Bleu),
        Violoniste#=JusDeFruit,
        voisin(Docteur,Renard),
        voisin(Diplomate,Cheval),

        labeling(Type,Vars).

voisin(X,Y):-
        (X#=Y+1 ; Y#=X+1).
```

**Exercice 2 : (coloriage de graphes)**
Ecrire un programme prolog permettant de calculer le nombre minimum de couleur permettant de colorier correctement le graphe suivant :



Un graphe est correctement colorié si deux nœuds adjacents dans le graphe ne reçoivent pas la même couleur. *Le programme suivant trouve un coloriage possible. Est-il possible de le modifier de manière à trouver la solution optimale en nombre de couleurs.*

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

coloriage(L):-
        L=[A,B,C,D,E,F]
        fd_domain(L,1,6),
        A#\=B, A#\=C, A#\=E, A#\=F, B#\=C, B#\=D, B#\=F,
        C#\=D, C#\=E, D#\=E, D#\=F, E#\=F,
        labeling([], L).
```

**Exercice 3 : (investissements)**
Un investisseur dispose de 14000 $ de réserve d'argent. Il souhaite investir son capital de manière à maximiser les gains ou NPV (Net Present Value). Le tableau suivant indique pour chaque investissement (investment), le budget nécessaire (cash required) et le gain espéré (NPV).

| investment | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| cash required | $5k | $7k | $4k | $3k | $4k | $6k |
| NPV | $16k | $22k | $12k | $8k | $11k | $19k |

cash = $14k

NPV = $42k

Il s'agit d'écrire un programme permettant de trouver la liste des investissements à faire pour maximiser ses gains.

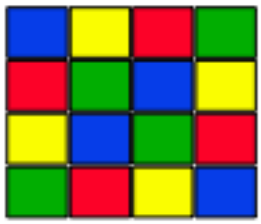
```
invest(L, Cout) :-
        L = [Use1, Use2, Use3, Use4, Use5, Use6],
        domain(L,0,1),
```

```
        5*Use1+7*Use2+4*Use3+3*Use4+4*Use5+6*Use6#<=14,
        Cout#= 16*Use1+22*Use2+12*Use3+8*Use4+11*Use5+19*Use6,
        maximize(labeling([], L), Cout).
```

**Exercice 4 : (Latin squares)**

Étant donnée n couleurs, un carré latin (ou quasigroup) d'ordre n est un carré n x n colorié tel que :

- toute cellule est coloriée
- chaque couleur apparaît exactement une fois sur chaque ligne
- chaque couleur apparaît exactement une fois sur chaque colonne



Latin squares d'ordre 4

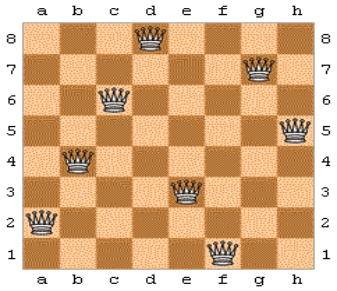
Écrire un programme permettant de résoudre ce problème.

```
:-use_module(library(clpfd)).
:-use_module(library(lists), [append/3]).

latinsquare(N) :-
  list(P,N),
  matrice(P,N),
  append_all(P,Vars),
  domain(Vars,1,N),
  contraintesLignes(P),
  contraintesColonnes(P,N),
  labeling([], Vars),
  afficherMatrice(P).


afficherMatrice([]).
afficherMatrice([L|M]):-
        afficherLigne(L),nl,
        afficherMatrice(M).

afficherLigne([]).
afficherLigne([X|L]):-
        write(X),write(' '),
        afficherLigne(L).


list([],0):-!.
list([_|L],N):-
        N1 is N-1,
        list(L,N1).
matrice([],_):-!.
matrice([L|M],N):-
        list(L,N),
        matrice(M,N).

contraintesLignes([]).
contraintesLignes([L|M]):-
        all_different(L),
        contraintesLignes(M).

contraintesColonnes(_,0).
contraintesColonnes(P,N):-
```

```
        variablesColonnes(P,N,C),
        all_different(C),
        N1 is N-1,
        contraintesColonnes(P,N1).

variablesColonnes([],_,[]).
variablesColonnes([L|M],N,[X|C]):-
        niemeVars(L,N,X),
        variablesColonnes(M,N,C).

niemeVars([X|_],1, X).
niemeVars([_|L],N, X):-
        N>1,
        N1 is N-1,
        niemeVars(L,N1,X).

append_all([], []).
append_all([P|R], X) :-
  append(P, Y, X),
  append_all(R, Y).
```

**Exercice 4 : (n-reines)**

Étant donné un échiquier de taille n x n, le problème de n-reines consiste à placer n reines sur un tel échiquier sans que deux reines en soient en prises : deux reines ne peuvent pas être sur la même ligne, colonne ou diagonale. Écrire le prédicat reines(N, L) qui résout ce problème où N est la taille de l'échiquier et L la liste des variables de taille N. Une variable Li de L indique que la reine de la ligne numéro i est placée sur la colonne Li.




8-reines

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

queens(N, L) :-
        list(N, L),
        domain(L,1,N),
        safe(L),
        labeling([],L).

safe([]).
safe([X| Y]) :-
        noattack(X, Y),
        safe(Y).

noattack(X, Xs) :-
        noattack(X, Xs, 1).

noattack(X, [], Nb) :- !.
noattack(X, [Y | Ys], Nb) :-
        X#\=Y,
        X#\= Y+Nb,
        X#\= Y-Nb,
```

```
        Nb1 is Nb+1,
        noattack(X, Ys, Nb1).

list(0, []) :- !.
list(N, [ _| L] ) :-
        M is N-1,
        list(M, L).
```

**Exercice 5 (sudoku)**

Le problème du Sudoku consiste à remplir la grille de sorte que chaque ligne, chaque colonne et chaque carré contiennent les chiffres 1 à 9. Exemple :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 9 | | | 1 | 6 | 2 | |
| 5 | 7 | | | 2 | 8 | | 3 | |
| 3 | | | 7 | | | | | 4 |
| 8 | 9 | | | 7 | | 4 | | |
| | 6 | | 5 | | 3 | | 9 | |
| | | 1 | | 9 | | | 7 | 6 |
| 6 | | | | | 7 | | | 8 |
| | 4 | | 1 | 3 | | | 6 | 5 |
| | 2 | 7 | 6 | | | 9 | | |

sudoku 9 x 9

Ecrire un programme en sicstus Prolog pour résoudre des sudoku. Vous écrirez un prédicat sudoku(L) qui réussit quand le sudoku avec les valeurs initiales données par L a une solution.

Deux solutions peuvent être envisagées :

1. La liste L est une liste de listes de variables
2. La liste L est une liste de lignes

Solution 1 :

```
sudoku :-
  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],
  domain(L,1,9),
  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]),
  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]),
   all_different([X11,X12,X13,X21,X22,X23,X31,X32,X33]),
   all_different([X14,X15,X16,X24,X25,X26,X34,X35,X36]),
   all_different([X17,X18,X19,X27,X28,X29,X37,X38,X39]),
   all_different([X41,X42,X43,X51,X52,X53,X61,X62,X63]),
   all_different([X44,X45,X46,X54,X55,X56,X64,X65,X66]),
   all_different([X47,X48,X49,X57,X58,X59,X67,X68,X69]),
   all_different([X71,X72,X73,X81,X82,X83,X91,X92,X93]),
   all_different([X74,X75,X76,X84,X85,X86,X94,X95,X96]),
   all_different([X77,X78,X79,X87,X88,X89,X97,X98,X99]),
   labeling([], L),
   write(X11), write(' '), write(X12), write(' '),write(X13), write('  '),
   write(X14), write(' '), write(X15), write(' '),write(X16), write('  '),
   write(X17), write(' '), write(X18), write(' '),write(X19), nl,
   write(X21), write(' '), write(X22), write(' '),write(X23), write('  '),
  write(X24), write(' '), write(X25), write(' '),write(X26), write('  '),
   write(X27), write(' '), write(X28), write(' '),write(X29), nl,
   write(X31), write(' '), write(X32), write(' '),write(X33), write('  '),
   write(X34), write(' '), write(X35), write(' '),write(X36), write('  '),
   write(X37), write(' '), write(X38), write(' '),write(X39), nl,
write(X41), write(' '), write(X42), write(' '),write(X43), write('  '),
   write(X44), write(' '), write(X45), write(' '),write(X46), write('  '),
   write(X47), write(' '), write(X48), write(' '),write(X49), nl,
write(X51), write(' '), write(X52), write(' '),write(X53), write('  '),
   write(X54), write(' '), write(X55), write(' '),write(X56), write('  '),
   write(X57), write(' '), write(X58), write(' '),write(X59), nl,
write(X61), write(' '), write(X62), write(' '),write(X63), write('  '),
   write(X64), write(' '), write(X65), write(' '),write(X66), write('  '),
   write(X67), write(' '), write(X68), write(' '),write(X69), nl,
write(X71), write(' '), write(X72), write(' '),write(X73), write('  '),
   write(X74), write(' '), write(X75), write(' '),write(X76), write('  '),
   write(X77), write(' '), write(X78), write(' '),write(X79), nl,
write(X81), write(' '), write(X82), write(' '),write(X83), write('  '),
   write(X84), write(' '), write(X85), write(' '),write(X86), write('  '),
   write(X87), write(' '), write(X88), write(' '),write(X89), nl,
write(X91), write(' '), write(X92), write(' '),write(X93), write('  '),
   write(X94), write(' '), write(X95), write(' '),write(X96), write('  '),
 write(X97), write(' '), write(X98), write(' '),write(X99), nl.
```

**Solution 2 :**

```
:- use_module(library(lists), [append/3]).
:- use_module(library(clpfd)).

sudoku(P) :-
  Rows = [R1,R2,R3,R4,R5,R6,R7,R8,R9],
  problem(P, Rows),
  append_all(Rows, Vars),
  domain(Vars, 1, 9),
  row_constraint(Rows),
  column_constraint(R1, R2, R3, R4, R5, R6, R7, R8, R9),
  block_constraint(R1, R2, R3),
  block_constraint(R4, R5, R6),
  block_constraint(R7, R8, R9),
  (labeling([ff], Vars) -> true),
```

```
  display_rows(Rows).

display_rows([]).
display_rows([[X1,X2,X3,X4,X5,X6,X7,X8,X9]|Rows]) :-
  format('~d ~d ~d ~d ~d ~d ~d ~d ~d \n', [X1,X2,X3,X4,X5,X6,X7,X8,X9]),
  display_rows(Rows).

row_constraint([]).
row_constraint([R|Rt]) :-
  all_distinct(R),
  row_constraint(Rt).

column_constraint([], [], [], [], [], [], [], [], []).
column_constraint([X1|R1], [X2|R2], [X3|R3], [X4|R4], [X5|R5], [X6|R6],[X7|R7], [X8|R8], [X9|R9]) :-
  all_distinct([X1,X2,X3,X4,X5,X6,X7,X8,X9]),
  column_constraint(R1, R2, R3, R4, R5, R6, R7, R8, R9).


block_constraint([], [], []).
block_constraint([X1,X2,X3|R1], [X4,X5,X6|R2], [X7,X8,X9|R3]) :-
  all_distinct([X1,X2,X3,X4,X5,X6,X7,X8,X9]),
  block_constraint(R1, R2, R3).

append_all([], []).
append_all([P|R], X) :-
  append(P, Y, X),
  append_all(R, Y).

problem(1, P) :- % shokyuu
P=[[1,_,_,8,_,4,_,_,_],
[_,2,_,_,_,_,4,5,6],
[_,_,3,2,_,5,_,_,_],
[_,_,_,4,_,_,8,_,5],
[7,8,9,_,5,_,_,_,_],
[_,_,_,_,_,6,2,_,3],
[8,_,1,_,_,_,7,_,_],
[_,_,_,1,2,3,_,8,_],
[2,_,5,_,_,_,_,_,9]].

problem(2, P) :- % shokyuu
P=[[_,_,2,_,3,_,1,_,_],
[_,4,_,_,_,_,_,3,_],
[1,_,5,_,_,_,_,8,2],
[_,_,_,2,_,_,6,5,_],
[9,_,_,_,8,7,_,_,3],
[_,_,_,_,4,_,_,_,_],
[8,_,_,_,7,_,_,_,4],
[_,9,3,1,_,_,_,6,_],
[_,_,7,_,6,_,5,_,_]].

problem(3, P) :- % chuukyuu
P=[[_,_,_,_,_,_,3,_,_],
[_,_,_,8,5,_,_,1,_],
[_,_,2,_,_,4,_,_,9],
[_,3,_,_,_,2,_,_,4],
[8,_,_,_,6,_,_,_,1],
[7,_,_,9,_,_,_,5,_],
[1,_,_,6,_,_,7,_,_],
```

```
[_,9,_,_,2,3,_,_,_],
[_,_,4,_,_,_,_,_,_]].

problem(4, P) :- % joukyuu
P=[[_,7,9,_,_,_,_,_,1],
[6,_,_,_,_,_,3,8,_],
[_,_,_,_,4,2,_,_,_],
[_,_,3,9,_,_,_,_,_],
[7,8,_,_,_,_,_,2,5],
[_,_,_,_,_,4,8,_,_],
[_,_,_,3,1,_,_,_,_],
[_,5,6,_,_,_,_,_,7],
[2,_,_,_,_,_,4,3,_]].

problem(5, P) :- % shokyuu; from Mr. Horai
P=[[_,5,_,7,_,1,_,4,_],
[7,_,3,_,_,_,1,_,2],
[_,8,_,4,_,6,_,9,_],
[9,_,4,_,6,_,8,_,3],
[_,_,_,8,_,7,_,_,_],
[1,_,8,_,5,_,6,_,9],
[_,1,_,6,_,3,_,8,_],
[5,_,6,_,_,_,7,_,1],
[_,3,_,5,_,9,_,2,_]].

problem(6, P) :- % Hard: suudoku2 99 (1989)
P=[[8,_,_,_,_,_,5,_,_,_],
[_,1,2,3,_,_,6,_,_],
[_,4,5,6,_,_,_,2,_],
[_,7,8,_,_,_,_,_,1],
[_,_,_,_,9,_,_,_,_],
[9,_,_,_,_,_,8,7,_],
[_,2,_,_,_,6,5,4,_],
[_,_,4,_,_,3,2,1,_],
[_,_,_,1,_,_,_,_,9]].
```