Contraintes sur des domaines finis en ECLIPSE CLP

Le but de ce TD/TP est de se familiariser avec les contraintes sur des domaines finis en utilisant ECLIPSE CLP et de comprendre son fonctionnement. Faire les exercices marqués difficiles à la fin.
Nous utilisons la libraire `ic`. Quelques prédicats pour les contraintes sur un domaine fini:

- `Vars #:: [+Integer1..+Integer2]` définit le domaine de la variables `Vars` ou de la liste des variables `Vars` d'être entre les deux bornes `Integer1` et `Integer2`.
- Les contraintes arithmétiques s'écrivent en utilisant les fonctions habituelles et les prédicats suivant: `#=`, `#\=`, `#<`, `#>`, `#<`, `#=<`, `#>=`
- `alldifferent(?ListeVars)`
  Ce prédicat décrit la contrainte qui impose que toutes les variables de la liste `?ListeVars` prennent des valeurs différentes.
- `labeling(?Vars)`
  Ce prédicat est utilisé pour rechercher des solutions des contraintes sur les variables `Vars`.
- `search(+L, ++Arg, ++Select, +Choice, ++Method, +Option)`
  permet de chercher une solution pour les variables `L` avec plein d'options (voir eclipseclp.org/doc/bips/lib/ic/search-6.html). Typiquement `Arg` est 0, `Select` est `first_fail`, `Choice` est `indomain_min`, `Method` est `complete` et `Option` est `[]`.

Un programme en ECLIPSE CLP pour résoudre une contrainte s'écrit en trois parties: définir les domaines des variables, décrire la contrainte, résoudre. Exemple d'un programme:

```
:- lib(ic).
probleme([X,Y,Z]) :- X #:: [0..5],[Y,Z] #:: [3..7], X+Y #< 2*Z, labeling([X,Y,Z]).
```

Ensuite on le compile et on interroge:

```
[eclipse]: probleme(L).
L = [0,3,3] ? ;
L = [0,3,4] ? ;
L = [0,3,5] ? ; etc.
```

ECLIPSE CLP utilise la borne consistance. Un peut demander par exemple:

```
[eclipse]: X #:: [0..5], [Y,Z] #::[3..7], X+2*Y #< 2*Z - 2.
X = X{0 .. 5}
Y = Y{3 .. 5}
Z = Z{5 .. 7}   ...
```

Exercice 1

- Rendez **sur papier** bornes consistantes la contrainte $X = 2 * Y + Z \wedge X + Y = 6$ avec domaines $X :: [2..7], Y :: |1..5], Z :: [1..2]$
- Comparez votre résultat avec celui de ECLIPSE CLP. Est-ce que cette contrainte a une solution ?

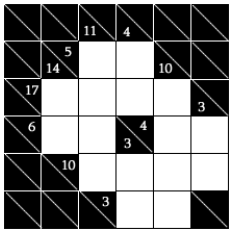
Exercice 2
On considère l'addition suivante :

```
  SEND
+ MORE
------
 MONEY
```

où chaque lettre représente un chiffre différent (compris entre 0 et 9). On souhaite connaître la valeur de chaque lettre, sachant que la première lettre de chaque mot représente un chiffre différent de 0. Pour modéliser ce problème on peut considérer que le mot `SEND` a comme valeur `1000*S + 100*E + 10*N + D`, etc. On peut aussi modéliser en utilisant des retenues.

- La variable M peut être contrainte d'avoir la valeur 1 en utilisant la borne consistance. Expliquez sur papier comment.
- Donnez un programme en ECLIPSE CLP qui résout ce problème.
- Vous pouvez utiliser le canevas qui se trouve dans le fichier moneycanvas.pl pour afficher la solution.

Exercice 3
Le Kakuro est un jeu. Le but est de remplir les cases blanches de la grille en utilisant seulement les chiffres de 1 a 9. Chaque Kakuro est composé de plusieurs bloc disposes sur des lignes et des colonnes, limités par des cases noires, de la même manière que les mots croisés. Ces cases doivent être remplies par des chiffres dont la somme correspond au nombre indiqué dans la case noire (en bas, pour le bloc vertical, en haut pour le bloc horizontal). Aucun chiffre ne doit apparaître deux fois dans le même bloc. Considérez la grille suivante:



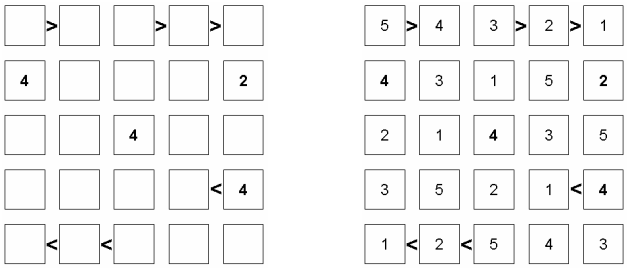
- Modélisez le problème comme un problème de satisfaction de contraintes (variables, leurs domaines, contraintes).
- Écrivez un programme en ECLIPSE CLP qui résout cette grille.
- **(difficile)** Écrivez un programme en ECLIPSE CLP qui résout une grille quelconque.

Exercice 4
Le problème du Sudoku consiste à remplir une grille 9x9 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		

- Écrire un programme en ECLIPSE CLP pour résoudre le sudoku de l'exemple.
- Écrire un programme en ECLIPSE CLP 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. La liste L est une liste de triples (ligne, colonne, valeur). Vous trouverez une définition de la liste de l'exemple ainsi qu'un deuxième exemple dans le fichier sudokuex.pl. Le premier exemple peut être résolu sans faire appel à `labeling` alors que le deuxième a besoin de `labeling`. Utilisez les prédicats `displayline` et `displayseperator` pour afficher la solution trouvée.

Exercice 5
Le Futoshiki est un jeu qui est basé sur une grille dans laquelle sont inscrits des nombres suivant quelques règles simples. Sur une grille de 5x5, les nombres de un à cinq doivent être placés dans chaque ligne et chaque colonne, sans aucune répétition. Les signes "plus grand que" ou "plus petit que" entre les cases sont des indices qui doivent obligatoirement être respectés. Voici un exemple de Futoshiki et sa solution.



- Modéliser le problème **de l'exemple** comme un problème de satisfaction de contraintes (Quelles sont les variables, leur domaines et les contraintes ?)
- Donnez un programme qui résout le Futoshiki **de l'exemple**.
- **(difficile)** Donnez un programme qui résout un Futoshiki **quelconque** de taille 5x5.

Exercice 6
(Examen 2020) On considère le jeu suivant.

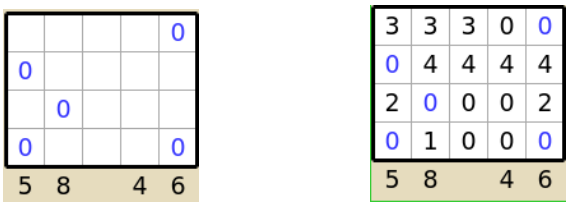


Figure 1: Une grille et sa solution

A chaque ligne de la grille on doit assigner un nombre N entre 1 et 4. Dans chaque ligne les nombres doivent être identiques ou égal à 0 et le nombre N doit apparaître exactement N fois. Une ligne ne peut pas contenir uniquement des 0. Les nombres dans la dernière ligne indiquent la somme des nombres dans les colonnes correspondantes.

- Écrivez un programme ECLiPSeCLP pour résoudre la grille de l'exemple.
  Indications: Mis à part le prédicat principal qui doit entre autres définir les variables et certaines contraintes, écrivez un prédicat `ligne` qui prend en entrée une liste de variables d'une ligne et qui génère les contraintes pour cette ligne: soit la première variable X est 0 et on traite récursivement les autres, soit elle n'est pas 0 et il doit y avoir le bon nombre d'autres variables avec la même valeur que X (ou 0). Pour cela utilisez un prédicat auxiliaire.

Exercice 7
(Examen 2013) On considère le jeu Bokkusu. Le but du jeu est de marquer en noir certaines cases de la grille (carré) donnée. Chaque case a deux valeurs : une valeur A et une valeur B. Les nombres à droite

dénotent les valeurs A des cases de chaque ligne correspondante et les nombres en bas dénotent les valeurs B des cases de chaque colonne correspondante (Ces valeurs vont toujours de 1 à la taille de la grille en augmentant de 1 de gauche à droite ou du haut vers le bas). Les valeurs à gauche indiquent la somme des valeurs B des cases noires pour chaque ligne et les valeurs en haut indiquent la somme des valeurs A des cases noires pour chaque colonne. Un exemple d'une grille et sa solution est donné dans la figure 2. Dans la solution de cette exemple on a par exemple $7 = 1 + 2 + 4$ (les cases avec valeur B de 1, 2 et 4 sont marquées noir) et $6 = 2 + 4$ (les cases avec valeurs A de 2 et 4 sont marquées).

Figure 2: Une grille Bokkusu et sa solution

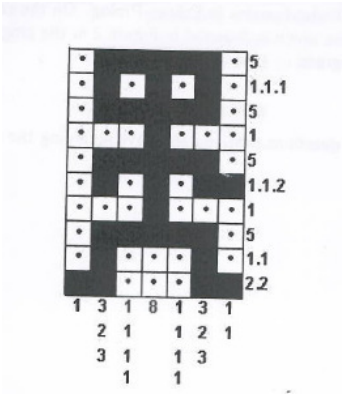
- Écrivez un programme en ECLIPSE CLP qui résout la grille de l'exemple.
- Écrivez un programme en ECLIPSE CLP qui résout une grille quelconque (la taille et les valeurs à gauche et en haut sont des paramètres).

Exercice 8

Programmer le problème des N reines en ECLIPSE CLP avec les contraintes sur domaine fini. Définir un prédicat `reines(N,L)` qui, quand N est un entier positif, donne en L les solutions du problème avec N reines. Indications: Programmer d'abord le problème des 4 reines. Comment générer les contraintes en général ? La contrainte qu'une reine n'est pas sur la même ligne qu'une autre est simple à exprimer. Pour les autres on peut par exemple (on peut évidement faire aussi autrement !) pour les contraintes diagonales tester chaque reine avec les autres (en commençant avec la première). On doit se rappeler de la distance en nombre de colonnes entre les deux reines à tester.

Exercice 9

((très) difficile) Le but du jeu des nonogrammes (voir Wikipedia) consiste à retrouver les cases noires dans chaque grille. Les chiffres donnés sur le côté et en haut de la grille vous donnent des indices. Ils indiquent la taille des blocs de cases noires de la ligne ou de la colonne sur laquelle ils se trouvent. Par exemple 3,4 à droite d'une ligne indique qu'il y a sur cette ligne, de gauche à droite, un bloc de 3 cases noires puis un bloc de 4 cases noires. Chaque grille résolue fait découvrir un dessin. Exemple d'une grille remplie:



- Écrire un programme qui résout un nonogramme quelconque. Celui de l'exemple est donné par

```
nonogramme(10,7,[[5],[1,1,1],[5],[1],[5],[1,1,2],[1],[5],[1,1],[2,2]],
                [[1],[3,2,3],[1,1,1,1],[8],[1,1,1,1],[3,2,3],[1,1]]).
```

  D'autres exemples se trouvent dans nonogramme.pl
  On peut commencer en donnant un programme spécifique pour le nonogramme le plus simple.