Contraintes sur des domaines finis en GNU Prolog

Le but de ce TD/TP est de se familiariser avec les contraintes sur des domaines finis en utilisant GNU Prolog.

GNU Prolog (Rappel)

GNU Prolog (`http://www.gprolog.org/`) est un Prolog avec un solutionneur intégré de contraintes sur un domaine fini. GNU Prolog se lance avec la commande `gprolog`. Quelques prédicats principaux additionnels pour les contraintes sur un domaine fini:

- `fd_domain(?Vars, +Integer1, +Integer2)` définit le domaine d'une variable `Vars` ou d'une liste de variables `Vars` d'être entre les deux bornes `Integer1` et `Integer2`.
- `fd_domain(?Vars,+ListeValeurs)` définit le domaine d'une variable ou d'une liste de variables `Vars` d'être la liste des valeurs dans `ListeValeurs`
- Les contraintes arithmétiques s'écrivent en utilisant les fonctions habituelles et les prédicats suivant: `#=`, `#\=`, `#<`, `#>`, `#<`, `#=<`, `#>=`
- `fd_all_different(?ListeVars)`
  Ce prédicat décrit la contrainte qui impose que toutes les variables de la liste `?ListeVars` prennent des valeurs différentes.
- `fd_labeling(?Vars, [])`
  Ce prédicat est utilisé pour rechercher des solutions des contraintes sur les variables `Vars`

Un programme en GNU Prolog pour résoudre une contrainte s'écrit en trois parties: définir les domaines des variables, décrire la contrainte, résoudre. Exemple:

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

Ensuite:

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

GNU Prolog utilise la borne consistance. Un peut demander par exemple:

```
| ?- fd_domain(X,0,5), fd_domain([Y,Z],3,7), X+2*Y #< 2*Z - 2.

X = _#3(0..5)
Y = _#25(3..5)
Z = _#47(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 avec le résultat de GNU Prolog.

Exercice 2

On considère les deux additions 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 utilisons des retenues.

- La variable M peut être contrainte d'avoir la valeur 1 en utilisant la borne consistance. Expliquez comment.
- Donnez un programme en GNU Prolog qui résout ce problème.
- Vous pouvez utiliser le canevas qui se trouve dans le fichier `~habermeh/clp/money.pl` pour afficher la solution.

Exercice 3

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		

Écrire un programme en GNU 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. La liste L est une liste de triples (ligne, colonne, valeur). Vous trouverez une définition de la liste de l'exemple sur le fichier `~habermeh/clp/sudoku.pl`. Utilisez le prédicat `displayline` pour afficher la solution trouvée.

Exercice 4

Programmer le problème des N reines en GNU Prolog 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 contraintes diagonales on teste 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.