Programmation ASP

Clingo. Ce TME utilise Clingo 4.5.4, la version vue en cours du solveur ASP de l'université de Potsdam, qui est l'un des mieux maintenus. C'est la composition de deux programmes : `gringo`, qui instancie le programme ASP, et `clasp`, qui résout le programme ASP propositionnalisé et calcule les answer sets. Le solveur `clasp` est basé sur un algorithme inspiré des solveurs SAT de type CDCL (mais avec quelques traitements spécifiques à l'ASP, notamment sur les agrégats). Il est téléchargeable à l'adresse suivante : `http://sourceforge.net/projects/potassco/files/clingo/4.5.4/`

Une fois le fichier `clingo-4.5.4-x86-linux.tar.gz` téléchargé, décompressez-le à l'endroit de votre choix, puis ouvrez un terminal et allez dans le repertoire où vous avez décompressé l'archive. Ce répertoire contient l'exécutable `clingo`.

Utilisation. Pour lancer clingo sur un programme asp, il faut lancer l'exécutable `clingo`. Le premier argument est le fichier d'entrée (en general un `.lp` ou `.sm`), qui doit contenir le programme ASP. Vous pouvez mettre plusieurs fichiers, auquel cas ils seront tous concaténés (par exemple si vous avez écrit votre programme en plusieurs modules). Ainsi pour obtenir les answer sets d'un fichier nommé `ex.lp`, il faut faire `./clingo ex.lp`.

Par défaut `clingo` ne calcule qu'un seul answer set. Pour les calculer tous, on ajoute un `0` en premier argument (`./clingo 0 ex.lp`). Pour en calculer n, on ajoute `n` (par exemple `./clingo 3 ex.lp`). Pour avoir toutes les options, taper (`./clingo 0 --help`).

Exercice 1 Premiers pas en ASP

Ce premier exercice est destiné à vous familiariser avec l'environnement de clingo en testant quelques programmes simples. Pour cela, donner le ou les ensembles-réponses correspondant aux programmes Π_1, Π_2, Π_3 et Π_4 :

$$\Pi_1 \left\{ \begin{array}{c} congés. \\ temps\_libre \leftarrow congés. \\ sortie\_ciné \leftarrow temps\_libre, congés. \end{array} \right. \quad \Pi_2 \left\{ \begin{array}{l} p \leftarrow not\ q. \\ q \leftarrow not\ r. \end{array} \right. \quad \Pi_3 \left\{ \begin{array}{c} p \leftarrow not\ q. \\ q \leftarrow not\ p. \\ r \leftarrow p. \\ r \leftarrow q. \end{array} \right. \quad \Pi_4 \left\{ \begin{array}{c} p \leftarrow not\ q. \\ q \leftarrow not\ p. \\ r \leftarrow not\ r. \\ r \leftarrow p. \end{array} \right.$$

Exercice 2 Problème des 8-reines

			×				
	×						
						×	
				×			
×							
							×
					×		
		×					

L'objectif est de résoudre grâce à l'ASP le problème des 8-reines. Pour cela, on procède par étape en répondant aux points suivants :

1. A l'aide d'une formule de choix, générer toutes les configurations possibles telles que deux reines ne peuvent se trouver sur la même ligne de l'échiquier.
2. Ajouter la contrainte qui interdit à deux reines de se trouver sur la même colonne de l'échiquier.
3. Ajouter la contrainte correspondant aux prises en diagonale.
4. Donner ensuite les solutions du problème des 8-reines dans le cas où une reine se trouve nécessairement aux coordonnées (1,1).

5. Donner enfin les solutions du problème des 8-Reines qui ne présentent aucune reine aux coordonnées (4,4) de l'échiquier.
6. Tester le programme avec d'autres tailles d'échiquier.

Exercice 3 Résolveur de Sudoku

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

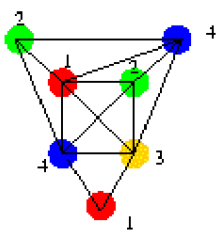
Sudoku

En 1979, un pigiste spécialisé dans les casse-têtes, **Howard Garns**, crée le premier jeu moderne de Sudoku. Une **grille de Sudoku** est un carré de neuf cases de côté, subdivisé en autant de carrés identiques, appelés régions (voir figure ci-contre). La **règle du jeu** est simple : chaque ligne, colonne et région ne doit contenir qu'une seule fois tous les chiffres de un à neuf. Formulé autrement, chacun de ces ensembles doit contenir tous les chiffres de un à neuf. Aujourd'hui, la renommée de ce jeu est telle que la plupart des journaux proposent une ou plusieurs grilles de Sudoku à leurs lecteurs.

Dans cet exercice, nous nous proposons d'écrire un programme en ASP qui permet de résoudre une grille de Sudoku telle que celle qui est présentée dans l'encadré ci-dessus. Pour cela, répondre aux questions suivantes :

1. Ecrire un programme permettant de générer toutes les grilles possibles sans prise en compte des trois contraintes liées au jeu de Sudoku. Les coordonnées utilisées sont des couples (x, y) où x et y correspondent à la case située à la ligne x et à la colonne y et prennent leurs valeurs dans l'intervalle $[0, 8]$. Par contre, les nombres entrés dans les grilles sont les entiers compris entre 1 et 9.
2. Limiter les grilles générées à celles présentant les valeurs de la grille proposée ci-dessus en ajoutant une liste de faits. Ainsi, la case $(0, 1)$ doit prendre la valeur 7 et la case $(1, 0)$ la valeur 8.
3. Traduire dans le langage de `smodels` les trois contraintes qui définissent le sudoku.
4. Chercher la ou les ensembles-réponses correspondant à la grille donnée ci-dessus.
5. Essayer le programme avec d'autres grilles et proposer une manière d'évaluer la difficulté de résolution d'une grille de Sudoku.

Exercice 4 Question de coloration



Un graphe G est dit *n-colorable* s'il existe une fonction f qui associe à chaque nœud de G un nombre entier compris entre 1 et n, telle que $f(x) \neq f(y)$ pour toute paire (x, y) de noeuds adjacents dans G.

Ecrire le programme permettant de trouver toutes les colorations possibles d'un graphe G.

Exercice 5 Casse-tête logique

On considère cinq maisons de cinq couleurs différentes. Dans chacune de ces maisons vit une personne d'une nationalité différente. Chacune de ces personnes boit une certaine boisson, fume une certaine marque de cigarettes et garde un certain type d'animal. Aucun d'entre eux ne possède le même animal qu'un autre, ne boit la même boisson ou ne fume la même marque de cigarettes. De plus, nous connaissons les informations suivantes :

1. Le Britannique vit dans la maison rouge.
2. Le Suédois garde un chien.
3. Le Danois boit du thé.
4. La maison verte est à gauche de la maison blanche.
5. Le propriétaire de la maison verte boit du café.
6. La personne qui fume des Pall Mall possède des oiseaux.
7. Le propriétaire de la maison jaune fume des Dunhill.
8. La personne qui vit dans la maison qui se trouve exactement au centre boit du lait.
9. Le Norvégien vit dans la première maison.
10. La personne qui fume des Blend vit à côté de celle qui possède des chats.
11. La personne qui possède des chevaux vit à côté du fumeur de Dunhill.
12. La personne qui fume des Bluemaster boit de la bière.
13. L'Allemand fume des Princess.
14. Le Norvégien vit à côté de la maison bleue.
15. La personne qui fume des Blend a un voisin qui boit de l'eau.

La question est alors : qui possède des poissons ?

Exercice 6 Organisation de championnat

Le but est de programmer la grille d'un match de championnat.

On considère qu'il y a n_e équipes participantes. Les matchs de championnat peuvent se dérouler le mercredi ou le dimanche. Le championnat dure (au maximum) n_s semaines (avec 2 matchs par semaine), soit $n_j = 2 * n_s$ jours de match.

Les matchs se font sur le terrain de l'une des deux équipes, qui est considérée comme jouant *à domicile*. L'autre équipe joue *à l'extérieur*. Etant donné les déplacements et la fatigue du match, chaque équipe ne peut jouer plus d'un match par jour. Sur la durée du championnat, chaque équipe doit rencontrer l'ensemble des autres équipes une fois à domicile et une fois à l'extérieur, soit exactement 2 matchs par équipe adverse.

Le but est de produire un planning des matchs, indiquant pour chaque jour quelles équipes s'affrontent en précisant où ont lieu les matchs.

Ecrire un programme ASP générant une grille de championnant tenant compte de ces contraintes.

Exercice 7 Organisation de championnat "équilibré"

On ajoute les contraintes suivantes au problème du championnat de l'exercice précédent.

— Matchs le dimanche.

1. Pour éviter de trop perturber les études des joueurs, on souhaite s'assurer que chaque équipe joue au moins p_{ext}% de ses matchs à l'extérieur des dimanches (par défaut on pose $p_{ext} = 50$).
2. De même, pour s'assurer que les supporter puissent assister à suffisamment de matchs, on veut que chaque équipe joue au moins p_{dom}% de ses matchs à domicile des dimanches (par défaut on pose $p_{dom} = 40$).

— Matchs consécutifs. On préfère alterner les matchs à domicile et à l'extérieur. On impose donc que :

1. Aucune équipe ne joue (strictement) plus de deux matchs consécutifs à l'extérieur.

2. Aucune équipe ne joue (strictement) plus de deux matchs consécutifs à domicile.

1. Ajouter au programme de l'exercice précédent des règles pour prendre en compte les contraintes d'équilibre.
2. Modifier le programme en utilisant les meta instructions d'optimisation (`#minimize`) pour déterminer automatiquement le nombre de jour minimal pour organiser le championnat.