Master d'Ingéniérie Informatique de Paris Diderot - Paris 7
M1: Prolog et programmation par contraintes

Examen partiel du 4 novembre 2011 - Durée: 1h45

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**3 points**)

Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :

1. `A=B,B=C,C=d.`
2. `A=B,(B=C;C=d).`
3. `(X is 1; X is 2), !, (Y is 3; Y is 4).`
4. `f(g(a),B)=f(g(B),A).`
5. `not(X>2), X is 1.`
6. `not(X==2).`

Exercice 2 (**6 points**)

Dans un graphe orienté, un *cycle* est une liste de sommets, tous différents, telle que chaque paire de sommets adjacents dans la liste soit reliée par un arc, et telle que le dernier sommet de la liste soit relié au premier par un arc.

Un graphe est donné par un certain nombre d'axiomes du prédicat `arc/2`, qui définissent à la fois les sommets et les arcs du graphe.

Par exemple, dans le graphe défini par le programme
`arc(a,b).arc(a,c).arc(b,c).arc(c,d).arc(d,a).`
la liste `[a,c,d]` est un cycle.

- Écrire un prédicat `est_cycle(+L)` qui réussit si la liste `L` est un cycle[1].

Un *cycle hamiltonien* est un cycle qui contient tous les sommets du graphe.

- À l'aide d'un prédicat `nombre_sommets(-N)` qui renvoie le nombre de sommets du graphe, et qu'il n'est pas nécessaire d'écrire, écrire un prédicat `est_cycle_hamiltonien(+L)`, qui réussit si L est un cycle hamiltonian. Vous pouvez utiliser le prédicat `length(+Liste,-Res)` qui calcule la longueur d'une liste.

On suppose que les sommets du graphe qui nous intéresse sont les termes `a`,`b`,`c`,`d`,`e`,`f`.

- À l'aide du prédicat `perm/2` vu en cours, qui produit toutes le permutations d'une liste donnée et qu'il n'est pas nécessaire d'écrire, écrire un prédicat `trouve_cycle_hamiltonien(-L)` qui renvoie les cycles hamiltonians du graphe, en utilisant un algorithme de type "générer et tester".

[1] Les prédicats auxiliaires utilisés devront être définis.

Exercice 3 (**6 points**)

Un nombre entier positif est *parfait* s'il est égale à la somme de ses diviseurs (propres, c.à.d. strictement plus petits que l'entier en question). Le premier nombre parfait est 6 = 1+2+3; le deuxième est 28 = 1+2+4+7+14.[2]

- Écrire un programme qui permet de vérifier si un nombre donné est parfait (le prédicat principale sera `est_parfait(+N)`). Vous pouvez utiliser l'opération binaire `mod` qui calcule le reste de la division entre deux entiers.

Exercice 4 (**5 points**)

Le *Jeu de Wythoff* est la variante suivante du jeu de 16 allumettes: la position de départ consiste en deux tas d'objets, et les coups disponibles pour les joueurs sont:

- Retirer un nombre quelconque d'objets d'un des deux tas.
- Retirer un nombre quelconque d'objets des deux tas (le même nombre d'objets doit être retiré de chaque tas).

1. Choisir une représentation des positions du jeu.
2. Définir, en fonction de la représentation choisie, un prédicat `move(+Position_courante, -Position_suivante)` qui implémente la règle du jeu.
3. Définir un prédicat `gagne(+position)` qui réussit si la position est gagnante.
4. Dessiner l'arbre de jeu dont la racine est la position où les deux tas contiennet 2 objets chacun, et MAX joue. Évaluer (à la main) cet arbre en utilisant l'algorithme minimax à arbre de jeu (on dira que la valeur d'une feuille est 1 si MAX gagne, 0 si MIN gagne).

[2] On ne sait pas si l'ensemble des nombres parfaits est fini ou infini.