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

Examen partiel du 5 novembre 2010 - Durée: 1h45

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**3 points**)

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

1. `[X|[Y|Z]]=[1,2,3].`
2. `(X=1;X=2),Y is X+1.`
3. `(X=1;X=2),Y=X+1.`
4. `(X=1;X=2),Y==X+1.`
5. `(X=1;X=2),!,Y is X+1.`
6. `(X=1;X=2),Y=:=X+1.`

Exercice 2 (**5 points**)

Soit `p` un prédicat à n arguments, défini par un certain nombre d'axiomes de la forme `p(t1,...,tn).` et de règles de la forme `p(t1,...,tn):-r1,..,rk.` (l'entier positif `k` pouvant varier d'une règle à l'autre).

On définit les prédicats *d'élagage préfixe*, `ppre`, et *d'élagage postfixe*, `ppost`, de `p` comme suit:

- pour tout axiome `p(t1,...,tn).` définissant `p`, on pose

  `ppre(t1,...,tn):-!.`

  et

  `ppost(t1,...,tn):-!.`

- Pour toute règle `p(t1,...,tn):-r1,..,rk.` définissant `p`, on pose

  `ppre(t1,...,tn):-!,r1,..,rk.`

  et

  `ppost (t1,...,tn):-r1,..,rk,!.`

Pour chacune des propositions suivantes, fournir une preuve ou un contrexemple, `t1`,...,`tn` étant des termes quelconques:

1. Si `p(t1,...,tn)` réussit, alors `ppost(t1,...,tn)` réussit.
2. Si `p(t1,...,tn)` réussit, alors `ppre(t1,...,tn)` réussit.
3. L'arbre de dérivation de `ppost(t1,...,tn)` possède au plus une branche succès.
4. L'arbre de dérivation de `ppre(t1,...,tn)` possède au plus une branche succès.

Exercice 3 (**7 points**)

- Principe du **tri insertion**: pour trier une liste `[X|L]` on commence par trier (par insertion) `L`, puis on insère `X` dans la liste ainsi triée, de telle sorte que le résultat demeure une liste triée.
- Principe du **tri fusion**: pour trier une liste `L` (d'au moins deux éléments) on commence par créer les listes `LI` des éléments d'indice impair de `L`, et `LP` de ses éléments d'indice pair. Esuite, on trie (par fusion) `LI` et `LP`. Finalement, on fusionne le deux listes triées ainsi obtenues.
- Principe du **tri à bulles**: une *itération du tri à bulles* sur `L` consiste en échanger chaque élément de `L` avec l'élément qui le suit dans `L`, à condition que le premier soit strictement plus grand que le deuxième, à partir de la paire composée par le premier et le deuxième élément de `L`. Le tri à bulle de `L` consiste en appliquer des itération du tri à bulles sur `L`, tant qu'au moins un échange a lieu.

Définir des prédicats[1]:

- `triinsertion(+liste_arg,-liste_res)` qui implémente le tri insertion d'une liste d'entiers. Prédicat auxiliaire suggéré: `insertion(+element,+liste_triee_arg,-liste_triee_res)`.
- `trifusion(+liste_arg,-liste_res)` qui implémente le tri fusion d'une liste d'entiers. Prédicats auxiliaires suggérés:

  `separation(+liste_arg, -liste_elts_impairs, -liste_elts_pairs)`

  `fusion(+liste_triee_arg1, +liste_triee_arg2, -liste_triee_res)`
- `tribulles(+liste_arg,-liste_res)` qui implémente le tri à bulles d'une liste d'entiers. Prédicat auxiliaire suggeré: `iteration(+liste_arg, -liste_res)`.

Comparer les complexités de vos implémentation de ces algorithmes de tri, en terme du *nombre de comparaisons d'entiers* nécessaires (dans le pire des cas) pour trier une liste de longueur n.

Exercice 4 (**5 points**)

Le *jeu de Grundy* est la variante suivante du jeu de 16 allumettes: la position de départ consiste en un unique tas d'objets (des allumettes ou des pions, par exemple), et le seul coup disponible pour les joueurs consiste à séparer un tas d'objets en deux tas de taille distincte. Les joueurs jouent à tour de rôle, jusqu'à ce que l'un d'entre eux ne puisse plus jouer. Le joueur qui ne peut plus jouer est le perdant.

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 un tas de 6 objets, 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).

[1]Ne pas utiliser de prédicats du module `lists`.