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

Examen partiel du 31 octobre 2006 - Durée: 1h45

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**6 points**)

1. Définir un prédicat `exp(+Base,+Exposant,-Resultat)` tel que si, `b` et `e` sont des entiers non négatifs, le but `exp(b,e,R)` produit comme résultat $\texttt{R} = \texttt{b}^{\texttt{e}}$. Par exemple:

```
   ?- exp(2,3,R).
R = 8 ? ;
no
```

   Votre définition de `exp` doit implementer l'algorithme suivant:

   - si l'exposant est 0, le résultat est 1;
   - sinon, si la base est 0, le résultat est 0;
   - sinon, le résultat s'obtient en multipliant la base par le résultat obtenu avec la même base et le prédecesseur de l'exposant.

2. Définir, en utilisant `exp`, un prédicat `tour_exp(+Base,+Hauteur,-Resultat)`, tel que, si `b` et `e` sont des entiers non négatifs, le but `tour_exp(b,e,R)` produit comme résultat: $\texttt{R} = \underbrace{b^{b^{\cdots^{b}}}}_{\texttt{(e+1) fois}}$, avec associativité gauche pour l'opération x^y. Par exemple

```
 ?- tour_exp(2,3,R).
R = 256 ? ;
no
```

3. Définir, en utilisant `exp`, un prédicat `monstre(+Base,+Hauteur,-Resultat)`, tel que, si `b` et `e` sont des entiers non négatifs, le but `monstre(b,e,R)` produit comme résultat: $\texttt{R} = \underbrace{b^{b^{\cdots^{b}}}}_{\texttt{(e+1) fois}}$, avec associativité droite pour l'opération x^y. Par exemple

```
?- monstre(2,3,R).
R = 65536 ? ;
no
```

Exercice 2 (**7 points**)

On considère le programme:

```
member(X,[X|_]):-!.
member(X,[_|T]):-member(X,T).

d([],[]).
d([X|L],R):-member(X,L),d(L,R).
d([X|L],[X|R]):-not(member(X,L)),d(L,R).
```

1. Que calcule le prédicat `d(+ListeArgument,-ListeResultat)`?

On considère les programmes:

```
d1([],[]).
d1([X|L],R):-member(X,L),d1(L,R).
d1([X|L],[X|R]):-d1(L,R).

d2([],[]).
d2([X|L],R):-member(X,L),!,d2(L,R).
d2([X|L],[X|R]):-d2(L,R).

d3([],[]).
d3([X|L],R):-!,member(X,L),d3(L,R).
d3([X|L],[X|R]):-d3(L,R).
```

1. Quels sont les programmes, parmi `d`,`d1`,`d2` et `d3`, qui ont la même sémantique déclarative? (motiver la réponse).

2. Quels sont les programmes, parmi `d`,`d1`,`d2` et `d3`, qui ont la même sémantique opérationnelle? (motiver la réponse).

Exercice 3 (**7 points**) Deux jouers J et O sont face à un tas de briques. Chaque joueur, alternativement, doit retirer une, deux ou trois briques du tas. Le joueur qui retire la dernière brique a perdu. Le jeu commence avec un tas de 77 briques. Un *état du jeu* est un entier qui représente le nombre de briques restantes. L'état initial du jeu est donc 77.

1. Définir un prédicat `move(+Etat,-NouvelEtat)` qui implémente la règle du jeu.

2. Définir un prédicat `gagne(+Etat)` qui réussit si le joueur qui joue à partir de l'état donné a une stratégie gagnante. Utiliser le prédicat `move` et la negation `not` pour définir `gagne`.

Un *arbre de jeu* est un arbre dont le noeuds sont des couples:

- état du jeu (un entier non négatif),
- joueur qui a la main (J ou O).

Par exemple, le couple $(77, J)$ représente le début d'une partie entamée par J. Un noeud dont l'état est n a autant de fils qu'il y a d'états accessibles par un coup à partir de n, c.à.d. 3 fils si $n > 2$, 2 fils si $n = 2$, 1 fils si $n = 1$, aucun fils si $n = 0$ (une feuille).

Un *arbre de jeu évalué* est un arbre de jeu dont les noeuds sont etiquetés par un entier qui évalue le noeud (du point de vue de J):

Les valeurs des noeuds se calculent à partir des feuilles:

- La valeur d'une feuille $(0, J)$ est 1 (J a gagné). La valeur d'une feuille $(0, O)$ est -1 (J a perdu).
- $(n + 1, J)$ vaut le maximum des valeurs de ses fils.
- $(n + 1, O)$ vaut le minimum des valeurs de ses fils.

1. Dessiner l'arbre de jeu évalué de racine $(4, J)$.

2. Quelle est la valeur du noeud $(77, J)$? Donner une nouvelle définition de `gagne(+Etat)`, sans utiliser `move` mais uniquement une propriété (arithmétique) de l'état.