Examen Partiel de
Prolog et programmation par contraintes

6 novembre 2009

Documents autorisés. Durée 1h45. Barème indicatif. Dans tous les exercices, vous pouvez définir et utiliser des prédicats auxiliaires, si cela vous semble utile.

Exercice 1 (2 points)

Pour chacun des buts suivants, dire quel est le résultat renvoyé par l'interpréteur Prolog (sans justifier la réponse):

1. `(!-> X==Y;true).`
2. `not(3+7==10), 3+7=:=10.`
3. `not(not(!)).`
4. `[X,Y,[a]]=[1,2|Z].`
5. `X is 7, Y = X, Y<7.`
6. `X is 7, Y =:= X, Y<7.`

Exercice 2 (6 points)

On suppose de disposer, sans avoir à l'écrire, d'un prédicat `est_un_mot(+Liste)` qui réussit si son argument est une liste de caractères qui, concaténes, forment un mot de la langue française.

1. Écrire un prédicat `anagrammes(+Liste,-Resultat)` tel que, si `liste` est une liste de caractères, le but `anagrammes(liste, X)` renvoie en `X` les permutations de `liste` qui sont des mots de la langue française.
2. Écrire un prédicat `tous_les_mots(+Liste,-Resultat)` tel que, si `liste` est une liste de caractéres, le but `tous_les_mots(liste, X)` renvoie en `X` les mots de la langue française, toujours sous la forme de listes de caractères, qu'on peut écrire avec les caractères contenus dans `liste`, sans les utiliser nécessairement tous.

Exercice 3 (6 points)

Nous allons considérer une version très simplifiée du jeu du taquin: il s'agit d'un solitaire, dont l'echiquer est une matrice 2×2 qui contient les entiers $1, 2, 3$, et une case vide. Á partir d'une position donné, on peut déplacer un "bloc", adjacent à la case vide, sur la case vide. Par exemple, les positions atteignables à partir de la position:

3	
2	1

sont

3	1
2	

e

	3
2	1

La position finale est

1	2
3	

1. Choisir une représentation des positions du jeu du taquin en prolog, et définir un prédicat `move(+Config,-Nouvelle_Config)` qui implemente la règle du jeu: `move(config,X)` renvoie en `X` les deux configurations atteignables à partir de `config`.

2. Ecrire un prédicat `peut_terminer(+Config)` qui réussit s'il existe une séquence de coups qui, à partir de la position représentée par `Config`, mène à la position finale (attention aux bouclages: il faut éviter de revisiter de configurations déjà visitées par le passé...).

Exercice 4 (6 points)

Dans cet exercice, on suppose que les listes manipulées par les différents prédicats sont des listes d'entiers.

1. Écrire un prédicat `somme(+Liste,-Resultat)` qui renvoie dans `Resultat` la somme des éléments de la liste `Liste`.

2. Sans utiliser le prédicat `somme`, écrire un prédicat

   `somme_acc(+Liste,+Accumulateur,-Resultat)` qui renvoie dans `Resultat` la somme des éléments de la liste `Liste` et de l'entier `Accumulateur`. Ce prédicat doit itérativement additioner le premier élément de la liste à l'accumulateur, et transférer le contenu de l'accumulateur dans le résultat une fois que la liste est vide.

3. Écrire des prédicats `produit(+Liste,-Resultat)` et

   `produit_acc(+Liste,+Accumulateur,-Resultat)` qui calculent le produit des éléments d'une liste, selon les mêmes principes que dans les points 1) et 2).

4. Les prédicats `somme_acc` et `produit_acc` fonctionnent selon le même principe: on itère une opération (somme et produit respectivement) à partir de la tête de la liste, et on garde les résultats partiels dans l'accumulateur, dont la valeur initiale est 0 pour la somme et 1 pour le produit. Nous allons à présent généraliser ce comportament pour des opérations quelquonques.

   Écrire un prédicat `iteration(+L,+Op,+Accumulateur,-Resultat)` tel que, si `operation(+X,+Y,-Z)` est un prédicat sur les entiers, dont l'élément neutre est `n`, le but `iteration(liste,operation,n,X)` donne en `X` le résultat de l'iteration de `operation` sur `liste`. Par exemple, si on définit

   ```
   somme(X,Y,Z):-Z is X+Y.
   ```

   on doit avoir que

   ```
   iteration ([1,2,3], somme, 0, R).
   ```

   donne comme résultat `R=6`.

   On peut utiliser le prédicat `=..` vu en cours.