M1 Informatique – Université de Paris

Programmation Logique et par Contraintes

Examen partiel du 22 octobre 2021 - Durée: 2 heures

Documents autorisés; le barème est donné à titre indicatif.

La lisibilité et les commentaires du code seront pris en compte dans l'évaluation.

Exercice 1 (**5 points**) Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier). Dans les cas où plusieurs résultats sont renvoyées par des retours en arrière succéssifs, les indiquer tous, séparées par de ";":

1. `X=1, (Y=2;Y=3),!.`
2. `(X=1;X=2),!,Y=3.`
3. `Q=..[member,X,[1,2]],Q.`
4. `[a,b,c,d]=[A|[B|[C|D]]].`
5. `not(not(!)).`
6. `X is 2, Y is 3, X+Y==5.`
7. `setof(X, X<3, L).`
8. `setof(X,(X=0;X=1;X=2),L).`
9. `X=1, not(X=2).`
10. `not(X=2), X=1.`

Exercice 2 (**10 points**) Une séquence de nombres naturels $s_0 \dots s_{n-1}$ est *autoréférentielle*, abrégé AR, si pour tout i allant de 0 à $n-1$, s_i est le nombre d'occurrences de l'entier i dans la séquence. Par exemple $s = 2, 0, 2, 0$ est une séquence AR de longueur 4, car $s_0 = 2$ et 0 apparait deux fois dans s, $s_1 = 0$ et 1 n'apparait pas dans s et ainsi de suite. Une autre séquence AR de longueur 4 est $1, 2, 1, 0$, et ces sont les seules séquences AR de longueur 4. Dans cet exercice, vous allez programmer en Prolog la génération exhaustive des séquences AR de longueur donnée. Les séquences seront représentée par des listes d'entiers.

Pour commencer, remarquons que les entiers qui composent une séquence AR de longueur n sont forcément strictement plus petit que n. Cette remarque est utilisée dans la phase de génération des séquences: il suffit de générer toutes les séquences de longueur n dont les éléménts sont inférieurs à n, puis de selectionner les séquences AR parmi celles-là.

Dans chacun des points de cet exercice, vous pouvez définir et utiliser des prédicats auxiliaires, et bien sûr utiliser les prédicats des points précédents, même si vous ne les avez pas définis.

1. Écrire un prédicat `inf(+n,-m)` qui donne tous les nombre naturels inférieurs à `n`, dans un ordre quelconque. Par exemple:

```
[eclipse 1]: inf(2,I).
I = 0
Yes (0.00s cpu, solution 1, maybe more) ? ;
I = 1
Yes (0.00s cpu, solution 2)
```

2. Écrire un prédicat `genere(+n,-liste)` qui donne toutes les listes de longueur `n` dont les éléments sont de nombres naturels inférieurs à `n`, dans un ordre quelconque. Par exemple:

```
[eclipse 2]: genere(2,L).
L = [0, 0]
Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [0, 1]
Yes (0.00s cpu, solution 2, maybe more) ? ;
L = [1, 0]
Yes (0.00s cpu, solution 3, maybe more) ? ;
L = [1, 1]
Yes (0.00s cpu, solution 4)
```

3. Écrire un prédicat `compte(+liste,+entier,-resultat)` qui donne le nombre d'occurrences de l'entier `entier` dans la liste `liste`. Par exemple:

```
[eclipse 3]: compte([1,2,1,0],1,N).
N = 2
Yes (0.00s cpu)
```

4. Écrire un prédicat `ar(+liste)` qui réussit si la liste est AR. Par exemple:

```
[eclipse 4]: ar([1,2,1,0]).
Yes (0.00s cpu)
[eclipse 5]: ar([1,1,1]).
No (0.00s cpu)
```

5. Écrire un prédicat `genar(+entier,-liste)` qui donne toutes les séquences AR de longueur `entier` par génération exhaustive. Par exemple:

```
[eclipse 6]: genar(4,L).
L = [1, 2, 1, 0]
Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [2, 0, 2, 0]
Yes (0.00s cpu, solution 2, maybe more) ? ;
No (0.00s cpu)
```

6. Pour générer plus efficacement les séquences AR, on utilise la remarque suivante: si $s_0 \dots s_{n-1}$ est AR, alors $s_0 + \dots + s_{n-1} = n$. Cette remarque est utilisée dans la phase de génération des séquences: il suffit de générer toutes les séquences de longueur n dont les éléménts sont inférieurs à n et dont la somme est n, puis de selectionner les séquences AR parmi celles-là.

   Écrire un prédicat `genere1(+n,-liste)` qui donne toutes les listes de longueur `n` dont les éléments sont de nombres naturels inférieurs à `n`, et telles que la somme des éléments de la liste est `n`. Par exemple:

```
[eclipse 7]: genere1(2,L).
L = [1, 1]
Yes (0.00s cpu, solution 1, maybe more) ? ;
No (0.00s cpu)
```

7. Pour générer encore plus efficacement les séquences AR, on utilise la remarque suivante: si $s_0 \dots s_{n-1}$ est AR, alors $\sum_{i=0}^{n-1} i \times s_i = n$. Écrire un prédicat `genere2(+n,-liste)` qui donne toutes les listes de longueur `n` dont les éléments sont de nombres naturels inférieurs à `n`, telles que la somme des éléments de la liste est `n` et telles que la somme des produits de chaque élément par son indice est `n`. Par exemple:

```
[eclipse 8]: genere2(3,L).
L = [1, 1, 1]
Yes (0.00s cpu, solution 1, maybe more) ? ;
No (0.00s cpu)
```

8. (bonus) Discuter le gain d'efficacité obtenu en passant de `genere` à `genere1` et de `genere1` à `genere2`, pour la génération exhaustive des séquences AR.

Exercice 3 (**5 points**)

1. Écrire un prédicat `printSum(+l1,+l2)` qui prend deux listes d'entiers et affiche les sommes des éléments de même rang des deux listes, une par ligne. Par exemple:

```
[eclipse 1]: printSum([1,2],[3,4]).
4
6
Yes (0.00s cpu)
```

   Pour afficher, utilisez le prédicat `println/1`. Si les deux listes n'ont pas la même longueur, `printSum` échoue.

2. Écrire un prédicat `printAppend(+l1,+l2)` qui prend deux listes de chaînes de caractères et affiche les concatenations des éléments de même rang des deux listes, une par ligne. Par exemple:

```
[eclipse 2]: printAppend(["a","b"],["c","d"]).
ac
bd
Yes (0.00s cpu)
```

Pour concatener deux chaînes de caractères, utilisez le prédicat `string_concat(+chaine1,+chaine2,-res)`. Pour afficher, utilisez le prédicat `println/1`. Si les deux listes n'ont pas la même longueur, `printAppend` échoue.

3. Écrire un prédicat `twoListIter(+op,+l1,+l2)` qui prend un nom de prédicat binaire et deux listes, et qui applique le prédicat aux éléments de même rang des deux listes. Si les deux listes n'ont pas la même longueur, `twoListIter` échoue. Par exemple, ayant défini:

```
psum(X,Y):- S is X+Y, writeln(S).
pconcat(X,Y):-string_concat(X,Y,S) writeln(S).
```

on doit avoir:

```
[eclipse 3]: twoListIter(psum,[1,2],[3,4]).
4
6
Yes (0.00s cpu)
[eclipse 4]: twoListIter(pconcat,["a","b"],["c","d"]).
ac
bd
Yes (0.00s cpu)
```