M1 Informatique – Université Paris Cité

Programmation Logique et par Contraintes

Examen partiel du 27 octobre 2022 - 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 plusiers résultats sont renvoyées par des retours en arrière successifs, les indiquer tous, séparées par de `;`

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

Exercice 2 (**8 points**) Pour cet exercice, un *multiensemble à n éléments sur un support à k éléments* est une liste triée de longueur n dont les éléments sont des entiers compris entre 1 et k. Dans un premier temps, vous allez implementer la génération exhaustive des multiensembles. Vous pouvez utiliser des prédicats auxiliaires.

1. Définir un prédicat `gen(+N,+K,-L)` qui génére dans un ordre quelconque toutes les listes `L` de longueur `N` dont les éléments sont des entiers compris entre 1 et `K`. Par exemple:

```
[eclipse 36]: gen(2,2,L).
L = [1, 1]
Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [1, 2]
Yes (0.00s cpu, solution 2, maybe more) ? ;
L = [2, 1]
Yes (0.00s cpu, solution 3, maybe more) ? ;
L = [2, 2]
Yes (0.00s cpu, solution 4)
```

2. Définir un prédicat `sorted(+L)` qui réussit si la liste d'entiers `L` est triée.
3. En utilisant `gen` et `sorted`, définir un prédicat `genex(+N,+K,-L)` qui génére dans un ordre quelconque tous les multiensembles à `N` éléments sur un support à `K` éléments. Par exemple:

```
[eclipse 10]: genex(2,3,L).
L = [1, 1]
Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [1, 2]
Yes (0.00s cpu, solution 2, maybe more) ? ;
L = [1, 3]
Yes (0.00s cpu, solution 3, maybe more) ? ;
L = [2, 2]
Yes (0.00s cpu, solution 4, maybe more) ? ;
L = [2, 3]
Yes (0.00s cpu, solution 5, maybe more) ? ;
```

```
L = [3, 3]
Yes (0.00s cpu, solution 6, maybe more) ? ;
No (0.00s cpu)
```

4. Combien y a-t-il de multiensemble de taille 10 sur un support à 2 éléments? Et combien y a-t-il de listes non triées de longueur 10 dont les éléments sont 1 ou 2? Que peut-on conclure sur l'approche par génération exhaustive des multiensembles?

5. Définir un prédicat `geneff(+N,+K,-L)` qui génére dans un ordre quelconque tous les multiensembles à `N` éléments sur un support à `K` éléments, par récurrence sur `N`. Les listes engendrées seront triées par construction, donc aucun test à postériori ne sera nécessaire.

Exercice 3 (**7 points**) **The silver dollar game** est le jeu combinatoire suivant:

- Position initiale: $k \geq 1$ pièces sont placées dans un tableau de taille $> k$, comme dans la figure (où $k = 3$):

- Règle du jeu: chaque joueur à son tour choisit une pièce et la déplace vers la gauche d'autant de cases qu'il veut, à condition de ne pas dépasser la case immédiatement à droite de la pièce suivante (ou la première case du tableau si la pièce déplacée est la plus à gauche). Donc l'unique position finale est celle où les pièces occupent les k premières cases les plus à gauche. Pour l'exemple ci-dessus, l'unique position finale est:

  Le joueur qui ne peut plus jouer, car la position finale a été atteinte, a perdu.

1. Dessiner l'arbre de jeu complet à partir de la position

   Cette position est-elle gagnante ou perdante?

2. Proposer une représentation en Prolog des positions de ce jeu, et définir un prédicat `move(+Pos1,-Pos2)` qui permet de jouer (remarque: une position est determinée par la liste des indices de ses pièces).

3. Définir le prédicat `gagne(+Pos)` qui réussit si et seulement si l'argument représente une position gagnante, en utilisant l'approche "force brute".

4. Ce jeu est en fait une variante du jeu de nim: étant donnée une position c à k pièces du silver dollar game il est possible de construire une position c' à $\lceil k/2 \rceil$ rangées du jeu de nim[1] de telle manière que c est gagnante si et seulement si c' est gagnante. Expliquer comment construire c' à partir de c, et utiliser cette traduction et votre connaissance du jeu de nim pour décider si la position initiale à trois pièces ci-dessus est gagnante ou perdante, et pour vérifier que votre réponse à la question 1. est correcte.

[1] Où $\lceil\ \rceil$ désigne la partie entière supérieure, et / la division entière. Il y aura 2 rangées pour l'exemple de position à 3 pièces ci-dessus, et une seule rangée pour l'exemple à 2 pièces.