M1 Informatique – Univ Paris Diderot

Programmation Logique et par Contraintes

Examen partiel du 17 octobre 2019 - Durée: 1h50

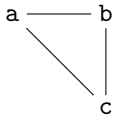
Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**5 points**) Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :

1. `X is 1+1 , X =:= 1+1.`
2. `X is 1+1 , X == 1+1.`
3. `X = 1+1 , X == 1+1.`
4. `Q =.. [member,1,[1]] , Q.`
5. `f( X , Y ) =.. L , member( Z , L ).`
6. `not( not( false ; true ) ).`
7. `not( not( ( ! , false ) ; true ) ).`
8. `bagof( X , ( member( X , [1,2,3,4,5] ) , X mod 2 =:= 1 ),L ).`
9. `bagof(X , ( member( X, [1,2,3,4,5] ) , ! ) , L ).`
10. `[X,a|[Y|Z]] = [1,DEUX,3,4,5].`

Exercice 2 (**10 points**) Dans cet exercice les différentes questions ne sont pas indépendantes. Toutefois, chaque question peut être résolue en supposant résolues les précédentes, même si elles n'ont pas été traitées.

Un graphe simple non dirigé est un couple (S, A), où S est l'ensemble de *sommets* et A est un ensemble de paires de sommets: les *arêtes*. Par exemple, $(\{\mathtt{a},\mathtt{b},\mathtt{c}\},\{\{\mathtt{a},\mathtt{b}\},\{\mathtt{a},\mathtt{c}\},\{\mathtt{b},\mathtt{c}\}\})$ est le graphe complet à trois sommets dessiné ci-dessous:



Le but de cet exercice est d'engendrer dans un premier temps tous les graphes simples non dirigés sur un ensemble de sommets donné, en utilisant la méthode de génération exhaustive.

Nous allons représenter les arêtes par des couples de sommets, et les graphes par la liste de leurs arêtes. Donc par exemple le graphe dessiné ci-dessus est représenté par la liste `[(a,b),(a,c),(b,c)]` (remarquez que l'ordre des arêtes dans la liste, tout comme l'ordre des sommets dans chaque couple représentant une arête, est sans importance. Remarquez aussi que les éventuels sommets isolés du graphe sont oubliés dans la représentation.).

1. Écrire un prédicat `all_arcs(+liste_sommets,-liste_aretes)`, tel que `all_arcs([`$\mathtt{s}_1$`,...,`$\mathtt{s}_n$`],A)` engendre la liste de tous les couples de sommets `(`$\mathtt{s}_i$`,`$\mathtt{s}_j$`)`, avec `i`$<$`j`. On aura par exemple:

   ```
   [eclipse 1]: all_arcs([a,b,c,d],A).
   A = [(a, b), (a, c), (a, d), (b, c), (b, d), (c, d)]
   Yes (0.00s cpu)
   ```

2. Écrire un prédicat `subsets(+set,-subset)`, tel que `subsets([`$\mathtt{s}_1$`,...,`$\mathtt{s}_n$`],S)` engendre les uns après les autres tous les sous-ensembles de $\{\mathtt{s}_1,\ldots,\mathtt{s}_n\}$, sous la forme de listes $[\mathtt{s}_{i_1},,\ldots,\mathtt{s}_{i_k}]$, avec $1 \leq \mathtt{i}_1 < \ldots < \mathtt{i}_k \leq \mathtt{n}$, et $\mathtt{k} \leq \mathtt{n}$. L'ordre dans lequel les sous-ensembles sont engendrés est sans importance. On aura par exemple:

   ```
   [eclipse 2]: subsets([a,b],S).
   S = [a, b]
   Yes (0.00s cpu, solution 1, maybe more) ? ;
   S = [a]
   Yes (0.00s cpu, solution 2, maybe more) ? ;
   S = [b]
   Yes (0.00s cpu, solution 3, maybe more) ? ;
   S = []
   Yes (0.00s cpu, solution 4)
   ```

3. Remarquer qu'un graphe sur un ensemble de sommets donné n'est rien d'autre qu'un sous-ensemble de l'ensemble de toutes les arêtes sur ces sommets. En utilisant `all_arcs`, `subsets` et `bagof`, écrire un prédicat `all_graphs(+liste_sommets,-graphes)` qui engendre la liste de tous les graphes simples non dirigés sur les sommets dans `+liste_sommets`. L'ordre est sans importance. On aura par exemple:

```
[eclipse 3]: all_graphs([a,b,c], G).
G = [
[(a, b), (a, c), (b, c)],
[(a, b), (a, c)],
[(a, b), (b, c)],
[(a, b)],
[(a, c), (b, c)],
[(a, c)],
[(b, c)],
[]
]
Yes (0.00s cpu)
```

Le *degré* d'un sommet `x` dans un graphe est le nombre d'arêtes auxquelles le sommet `x` appartient. Dans le graphe complet à trois sommets, par exemple, le degré de chaque sommet est 2, tandis que dans le graphe $(\{\mathtt{a},\mathtt{b},\mathtt{c}\},\{\{\mathtt{a},\mathtt{b}\},\{\mathtt{b},\mathtt{c}\}\})$, le dégré de `a` et `c` est 1 et le degré de `b` est 2.

4. Écrire un prédicat `deg(+graphe,+sommet,-degre)` qui calcule le degré du sommet `sommet` dans le graphe `graphe`, et échoue si le sommet n'appartient pas au graphe. On aura par exemple:

   ```
   [eclipse 4]: degre([(a,c),(b,c)],c,N).
   N = 2
   Yes (0.00s cpu)
   [eclipse 5]: degre([(a,c),(b,c)],d,N).
   No (0.00s cpu)
   ```

5. Écrire un prédicat `vertex_list(+graphe,-sommets)`, qui engendre la liste sans répétitions des sommets d'un graphe. L'ordre dans lequel les sommets sont listés est sans importance. On aura par exemple:

   ```
   [eclipse 5]: vertex_list([(a,b),(a,c),(b,c),(c,d)],L).
   L = [a, b, c, d]
   Yes (0.00s cpu)
   ```

On dira qu'un graphe simple non dirigé est *pseudo-eulérien* si le degré de chacun de ses sommets est un nombre pair.

6. Écrire un prédicat `eulerian(+graphe)`, qui réussit si le graphe `graphe` est pseudo-eulérien. On aura par exemple:

   ```
   [eclipse 6]: eulerian([(a,b),(a,c),(b,c)]).
   Yes (0.00s cpu)
   [eclipse 12]: eulerian([(a,b),(a,c)]).
   No (0.00s cpu)
   ```

7. Écrire un prédicat `all_eulerians(+sommets,-graphes)`, qui engendre la liste de tous les graphes pseudo-eulériens sur les sommets donnés. On aura par exemple:

   ```
   [eclipse 7]: all_eulerians([a,b,c],E).
   E = [[(a, b), (a, c), (b, c)], []]
   Yes (0.00s cpu)
   ```

Exercice 3 (**5 points**) On considère le jeu soustractif $\{3,4\}$: une position est un nombre naturel, et les position atteignables en un coup à partir de $n \in \mathbb{N}$ sont $n-3$, à condition que $n \geq 3$, et $n-4$, à condition que $n \geq 4$. Donc par exemple les positions atteignables en un coup à partir de 7 sont 4 et 3, tandis que 2 est une position finale. Le joueur qui ne peut plus jouer perd.

1. Définir le prédicat `move(+position_courante,-nouvelle_position)` qui engendre les positions atteignables en un coup à partir de `position_courante`.

2. Définir le prédicat `gagne(+position)` utilisant la méthode "force brute", et dessiner l'arbre de dérivation de `gagne(3)`.

3. Définir un prédicat `gagne_bis(+position)` "intelligent" (= non récursif) pour ce jeu (pour cela, il peut être utile d'ébaucher la construction des ensembles P et G des positions perdantes et gagnantes).