Introduction à la programmation récursive

Répétition 3

12 novembre 2014

Correction exercices proposés

Exercice 1. Définir un prédicat `swap(+A, +B, +Ls, -Zs)`, vrai si la liste `Zs` est la liste `Ls` dans laquelle toutes les occurrences de `A` ont été remplacées par `B` et toutes les occurrences de `B` ont été remplacées par `A`.

Exercice 2. Définir le prédicat `skipN(+N, +M, +Xs, -Ys)` vrai si `Ys` est la liste `Xs` privée de `N` éléments sur `M` (on supprime les N premiers éléments).

Exercice 3. Définir le prédicat `perfect(+N)` vrai si et seulement si `N` est un nombre parfait. Pour rappel, un nombre est parfait si la somme de ses diviseurs (excepté lui-même) rend le nombre.

Exercice 4. Les nombres de Motzkin vérifient notamment les égalités suivantes :

$$M_0 \;=\; M_1 \;=\; 1$$

$$\forall n > 1 : M_n \;=\; \frac{3(n-1)M_{n-2} \;+\; (2n+1)M_{n-1}}{n+2}$$

Écrire une fonction `motz` qui à tout entier naturel n associe M_n.

Exercice 5. Soit $f : \mathbb{N} \to \mathbb{N}$, la fonction définie par :

$$f(n) \;=\; \sum_{i=0}^{n-1} (f(i) + 2 * i) \bmod (n + i + 3)$$

Définir un prédicat `f(+N, -R)` vrai si f(`N`) = `R`.

Exercices sur les ensembles

Exercice 6. Définir un prédicat `partition(+Es, -Pss)`, vrai si `Pss` est une partition de l'ensemble `Es`. Un ensemble est une liste sans répétition.

Exercice 7. Définir un prédicat `union(+As, +Bs, -Us)`, vrai si `Us` est l'union des deux ensembles `As` et `Bs`.

Exercices sur les arbres binaires complets

On considère des arbres dont seules les feuilles sont étiquetées et où chaque nœud a exactement 2 ou 0 fils. On représente les feuilles de l'arbre par leur étiquette et les nœuds internes par une paire pointée `[L|R]` dont le premier élément `L` est la représentation du sous-arbre de gauche du nœud et le deuxième élément `R` la représentation du sous-arbre de droite.

Exercice 8. Définir un prédicat `is_binTree(+Tr)`, vrai si et seulement si `Tr` est un arbre binaire complet.

Exercice 9. Définir un prédicat `count_leaves(+Tr, -N)`, où `N` est un naturel et `Tr` un arbre binaire complet, vrai si `Tr` contient exactement `N` feuilles.

Exercice 10. Définir un prédicat `depth_tree(+Tr, -N)`, où `Tr` est un arbre binaire complet et `N` un naturel, vrai si et seulement si l'arbre `Tr` a une profondeur égale à `N`. Par convention, la racine de l'arbre est située à une profondeur de zéro.

Exercice 11. Définir un prédicat `parcours(+Tr,-Ls)`, où `Tr` est un arbre binaire complet dont les feuilles sont étiquetées par des naturels et `Ls` est une liste, vrai si et seulement si `Ls` est la liste des feuilles rencontrées lors d'un parcours "droite - gauche" de l'arbre, mais en remplaçant les feuilles étiquetées d'un nombre impair par le nombre pair directement supérieur.

Exercice 12. Définir le prédicat `same_frame(+Tr1, +Tr2)` vrai si et seulement si les arbres binaires complets `Tr1` et `Tr2` ont le même ensemble de feuilles (dans un premier temps, avec le même nombre d'occurrences et, dans un second temps, sans cette contrainte).

Exercice 13. Définir le prédicat `simplify(+Tr1, -Tr2)` vrai si et seulement si `Tr1` et `Tr2` sont des arbres binaires complets dont les feuilles sont étiquetées par des naturels et que `Tr2` est la version simplifiée de `Tr1` obtenue en remplaçant les sous-arbres ayant deux feuilles étiquetées par le même naturel par ce naturel. Dans un premier temps, on ne simplifiera qu'à un niveau de l'arbre. Puis on simplifiera récursivement tant que les étiquettes obtenues coïncident.

Exercices sur les polynômes

Exercice 14. Écrire un prédicat `poly_add(+P1,+P2,-P3)` vrai si le polynôme `P3` est la somme des polynômes `P1` et `P2`.

Exercice 15. Écrire un prédicat `poly_mul(+P1,+P2,-P3)` vrai si le polynôme `P3` est le produit des polynômes `P1` et `P2`.

Exercice 16. Écrire un prédicat `poly_eval(+P, +X, -Y))` vrai si `P` est un polynôme et si `Y = P(X)`.

Exercice

Exercice 17. Écrire un prédicat `countdown(+Ns, +K, -Lo)` vrai si `Ns` est une liste de nombres naturels à partir de laquelle on peut obtenir le nombre `K`, en effectuant la suite d'opérations arithmétiques spécifiées par `Lo`.

```
?- countdown([4, 75, 10, 7, 25, 1], 405, Lo).

    Lo = [[7, +, 25, =, 32], [32, +, 1, =, 33],
          [33, *, 10, =, 330], [330, +, 75, =, 405]]
```

Exercices proposés

Exercice 18. Définir un prédicat `swap_depth(+A, +B, +Ls, -Zs)`, vrai si la liste `Zs` est la liste `Ls` dans laquelle toutes les occurrences de `A` ont été remplacées par `B` et toutes les occurrences de `B` ont été remplacées par `A` en profondeur.

```
?- swap_depth(a, s, [x, [a, b, s, [h, a]], [[[a], [b]], a]], X).

    X = [x, [s, b, a, [h, s]], [[[s], [b]], s]] ;
    false.
```

Exercice 19. Écrire un prédicat calculant le fonction :

$$f(n) = \sum_{i=0}^{n-1}((f(i) + f(n-i-1) + i^2) \ mod \ (10 + n + i))$$

Exercice 20. Définir un prédicat `inter(+As,+Bs,-Is)`, vrai si `Is` est l'intersection des deux ensembles `As` et `Bs`.

Exercice 21. Définir un prédicat `unionList(+Ess,-Us)` vrai si `Us` est l'union de tous les ensembles de la liste d'ensembles `Ess`.

Exercice 22. Définir un prédicat `interlist(+Ess,-Is)` vrai si `Is` est l'intersection de tous les ensembles de la liste d'ensembles `Ess`.

Exercice 23. Définir un prédicat `getset(+Ess,-Es)` vrai si `Es` est l'ensemble des éléments se trouvant dans un seul des ensembles contenu dans la liste d'ensembles `Ess`.

Exercice 24. Un arbre quelconque complètement étiqueté est un arbre dans lequel chaque nœud possède une étiquette et a un nombre quelconque de fils (de 0 à N). Choisissez une représentation pour ce type d'arbre.

Définir un prédicat `count_nodes(+Tr, -N)`, où `N` est un naturel et `Tr` un arbre quelconque complètement étiqueté, vrai si `Tr` contient exactement `N` nœuds.

Exercice 25. Définir un prédicat `depth_first(+Tr, -Ls)`, où `Tr` est un arbre quelconque complètement étiqueté et `Ls` est une liste, vrai si et seulement si `Ls` est la liste des étiquettes des nœuds rencontrés lors d'un parcours en profondeur d'abord, et de gauche à droite, de l'arbre.

Exercice 26. Écrire un prédicat `poly_comp(+P1,+P2,-P3)` vrai si le polynôme `P3` est la composition du polynôme `P1` par le polynôme `P2`.

Exercice 27. Écrire un prédicat `deriv_poly(+P1, -P2)` vrai si le polynôme `P2` est la dérivée première du polynôme `P1`.

Exercice 28. Définir un prédicat `quick(+As, -Bs)`, où `As` et `Bs` sont des listes de naturels, vrai si et seulement si `Bs` est la version triée dans l'ordre croissant de `As`. On demande que la recherche de la liste triée `Bs` soit effectuée par la méthode "Quicksort".

Exercice 29. On dispose d'un ensemble de coupures (pièces d'un euro, de 2, billets de 5, 10, 20, 50, 100, 200, 500) et on demande de combien de manière différentes il est possible de payer une certaine somme. Définir un prédicat permettant de calculer ce nombre.