Travaux Pratiques de calcul formel n°7

Cours d'Informatique de Deuxième Année

—DEUG Sciences S4-MI—

Algorithmique en MuPAD

L'objectif de ce TP est d'utiliser un logiciel de calcul formel, en l'occurence MuPAD, dans le but de programmer quelques algorithmes d'énumération et de manipulation de listes.

▸ **Exercice 1.** ***(Énumération d'arbres)*** *Nous considérons l'ensemble des arbres planaires enracinés. On note $D(z)$ la série formelle dont le coefficient de z^n est le nombre d'arbres planaires enracinés à n arêtes. $D(z)$ est solution de l'équation suivante :*

$$D(z) = 1 + z\,D(z)^2.$$

1. *À l'aide de MuPAD, résoudre cette équation. Réaliser un développement limité en 0 des deux solutions posssibles de l'équation. Conclure en donnant $D(z)$.*
2. *Il est possible de calculer les premiers termes de $D(z)$ sans résoudre l'équation précédente. Pour cela, nous procèderons en deux étapes :*

   écrire une fonction qui tronque un polynôme à un certain degré ;

   l'équation est en fait une équation de récurrence. Écrire une fonction implémentant directement cette récurrence, et calculant donc les n premiers termes de la série $D(z)$.

3. *Il est plus efficace de calculer directement le nombre d'arbres planaires enracinés à n arêtes plutôt que de calculer tous les premiers termes de la série $D(z)$. Implémenter cette récurrence. Tester avec l'option* `remember`*. Que se passe-t-il ?*

▸ **Exercice 2.** ***(Génération d'arbres)*** *Les arbres planaires enracinés peuvent être représentés par une liste de trois éléments $[a, A_1, A_2]$, où a représente l'arête pointée, A_1 et A_2 les sous-arbres respectivement avant et après cette arête. La liste vide [ ] est l'arbre réduit à un sommet.*

1. *Donner la représentation sous-forme de liste des arbres enracinés à 0, 1, 2 et 3 arêtes.*
2. *Écrire une fonction (sur le même modèle que les fonctions d'énumération précédentes) qui génère tous les arbres à n arêtes. Vérifier leur nombre avec la fonction* `nops`*.*

▸ **Exercice 3.** ***(Bons parenthésages)*** *On considère cette fois des listes de n éléments sur un alphabet de deux lettres (par exemple '(' et ')'). Une telle liste L d'éléments est un bon parenthésage si et seulement si $L = [\ (L_1)\ L_2\ ]$, où $[L_1]$ et $[L_2]$ sont des listes bien parenthésées.*

1. *Écrire une fonction récursive qui renvoie vrai si une liste passée en argument est un bon parenthésage, faux sinon.*
2. *Écrire une fonction qui génère toutes les permutations des éléments d'une liste. En déduire une fonction qui génère tous les bons parenthésages à n parenthèses. Quelle est l'efficacité de cette méthode ?*
3. *Écrire une fonction récursive appliquant de manière directe la définition d'un bon parenthésage pour générer (de manière efficace) tous les bons parenthésages à n parenthèses. Quel est le nombre de bons parenthésages à n parenthèses ?*