Programmation fonctionnelle TP 9

antoine.frenoy@inserm.fr
http://perso.crans.org/frenoy/caml2013.html

Jeudi 28 mars 2013

1 Des légumes et des types

Un commerçant vend des patates, des carottes et des navets. Les patates sont à 1 euro le kilo, les carottes à 1.30 euro le kilo, et les navets à 1.70 euro le kilo. Les clients se servent eux-même dans des sachets plastiques. Lors de l'encaissement, le prix de chaque sachet est déterminé par son poids (un nombre réel) et le type de légume qu'il contient (un seul type de légume par sachet). Le panier d'un client contient plusieurs sachets (éventuellement plusieurs sachets d'un même type de légume).

1. Définissez un type `legume`.
2. Définissez un type `sachet`.
3. Définissez un type `panier`.
4. Définissez une fonction `prixsachet` prenant en argument un élément de type `sachet` et retournant son prix en euros.
5. Définissez une fonction `prixpanier` prenant en argument un élément de type `panier` et retournant son prix en euros.

2 Monnaie

Le même commerçant doit rendre la monnaie à ses clients. Il utilise pour cela des pièces de 1, 2, 5, 10, 20, et 50 centimes, et de 1 et 2 euros.

1. Définissez un type `piece`.
2. Définissez une fonction `rendumonnaie` qui prend en argument un flottant (l'argent à rendre) et qui renvoie une liste de pièces optimale (utilisation du moins de pièces possible).

3 Nombres complexes

1. Définissez un type enregistrement permettant de représenter un nombre complexe en coordonnées cartésiennes.
2. Définissez un type enregistrement permettant de représenter un nombre complexe en coordonnées polaires.
3. Écrivez une fonction permettant de passez des coordonnées cartésiennes vers les coordonnées polaires.
4. Écrivez une fonction permettant de passez des coordonnées polaires vers les coordonnées cartésiennes.
5. Écrivez des fonctions pour additionner et soustraire deux nombres complexes en coordonnées carthésiennes.
6. Écrivez une fonction prenant en arguments un réel a, un nombre complexe en coordonnées cartésiennes x, un nombre complexe en coordonnées cartésiennes y, et renvoyant, en coordonnées cartésiennes, l'image de x par une rotation de centre y et d'angle a.

4 Expressions booléennes

On s'intéresse à des expressions logiques booléennes, qui sont composées des constantes "VRAI" et "FAUX", de l'opérateur unaire "NON", des opérateurs binaires "ET", "OU", "OU EXCLUSIF" et "IMPLIQUE".

1. Définissez un type `expression`.
2. Utilisez ce type pour représenter l'expression $(vrai \wedge faux) \vee (faux \Rightarrow (vrai \Rightarrow faux))$.
3. Définissez une fonction récursive `evalue` qui prend un argument de type `expression` et renvoie un élément de type `bool` indiquant si l'expression booléenne est vraie.

5 Tours de Hanoï

On dispose de trois tiges sur lesquelles on empile des disques de diamètres deux à deux distincts. On peut déplacer un disque d'une pile à une autre, avec les règles de déplacement suivantes :

- On ne peut déplacer qu'un disque à la fois.
- On ne peut prendre un disque que s'il est en haut de sa pile.
- On ne peut poser un disque que sur un disque (strictement) plus grand que lui ou sur une pile vide.

On part d'une position dans laquelle les n disques sont empilés sur la tige n°1 du plus grand au plus petit (le plus grand en dessous, le plus petit au dessus). On cherche à arriver en un nombre minimal de déplacements à une position dans laquelle les n disques sont empilés sur la tige n°3 du plus grand au plus petit.

Pour mieux comprendre, voici un exemple dans le cas où $n = 3$ disques :

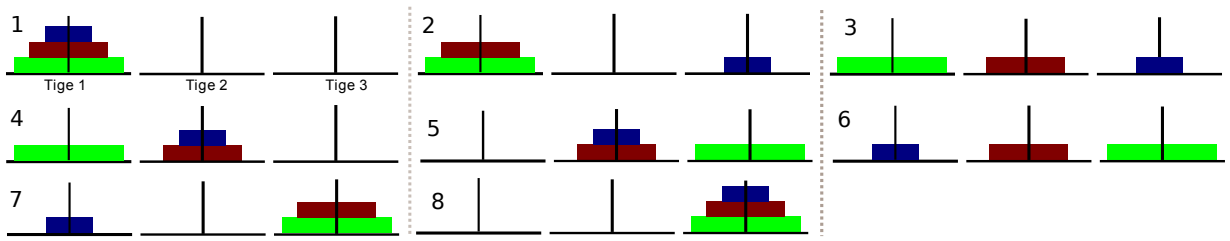

FIGURE 1 – Tours de Hanoi : exemple avec trois disques

1. Définir un type `tige` permettant de représenter les trois tiges.
2. Puisqu'on ne peut déplacer que le disque en haut d'une pile, pour représenter un mouvement il suffit de préciser la tige de départ et la tige de destination. Définir un type `mouvement`.
3. Écrivez une fonction `hanoi` prenant un argument `n` et renvoyant une liste de mouvements qui résout le problème dans le cas où il y a `n` disques.

   Indice : *Écrivez une fonction récursive* `hanoi_g` *plus générale, qui prend en arguments* `x` *(de type* `int`*),* `i`*,* `j` *et* `k` *(de type* `tige`*), et qui renvoie la liste des mouvements nécessaires pour déplacer les* `x` *disques les plus en haut de la pile* `i` *vers le haut de la pile* `j` *(en conservant leur ordre), où* `k` *est la pile restante (la pile qui n'est ni* `i` *ni* `j`*).*

6 Polynômes

Un monôme est représenté par un réel (le coefficient) et un entier naturel (le degré). Un polynôme peut alors être représenté par une liste de monômes. Pour cet exercice, nous déciderons que cette liste de monômes sera ordonnée du plus grand vers le plus petit degré.

- Écrivez un type `monome` (utilisez un enregistrement) et un type `polynome`.
- Quel est l'avantage de cette représentation par rapport à celle consistant à donner la liste de tous les coefficients même nuls ?
- Écrivez une fonction `evalpol` prenant en argument un polynôme `P` et un flottant `a` et renvoyant l'évaluation de `P` en `a`.

– Écrivez une fonction `somme` prenant en arguments deux polynômes et renvoyant leur somme.
– Écrivez une fonction `produit` prenant en arguments deux polynômes et renvoyant leur produit.
– Écrivez une fonction `derive` prenant en argument un polynôme et renvoyant sa dérivée.

7 Arbres binaires

On s'intéresse à un objet mathématique appelé "arbre binaire". Un arbre est un ensemble de données (dans notre cas des entiers) hierarchisées : chaque donnée est stockée dans un noeud, les noeuds sont reliés par des liens de parenté (le noeud le plus haut est appelé la racine, il a des fils qui ont eux même des fils et ainsi de suite). Pour plus d'informations, voir `http://fr.wikipedia.org/wiki/Arbre_binaire`.

Les arbres binaires peuvent être définis récursivement : un arbre binaire est soit l'arbre vide, soit constitué d'un noeud (associé à un entier) avec deux fils (le fils "gauche" et le fils "droit", chacun de ces fils étant lui-même un arbre binaire).

1. Définissez un type `arbrebinaire`.
2. Définissez une fonction `taille` prenant en argument un arbre binaire et renvoyant son nombre de noeuds.
3. Définissez une fonction `hauteur` prenant en argument un arbre binaire et renvoyant sa hauteur, c'est à dire la distance entre la racine et son descendant le plus éloigné.
4. Définissez une fonction `appartient` prenant en arguments un arbre binaire et un entier et indiquant si l'entier apparait dans l'arbre binaire.
5. Définissez une fonction `getdata` prenant en argument un arbre et renvoyant la liste des entiers associés aux noeuds de cet arbre.
6. Définissez une fonction `tousp` prenant en arguments un arbre et un prédicat et vérifiant que tous les entiers associés aux noeuds de l'arbre vérifient le prédicat.
7. Définissez une fonction `unp` prenant en arguments un arbre et un prédicat et vérifiant qu'au moins un des entiers associés aux noeuds de l'arbre vérifie le prédicat.

8 Arbres binaires de recherche

Les arbres binaires de recherche sont des arbres binaires vérifiant la propriété suivante : pour tout noeud, l'entier qui lui est associé est supérieur à tout les entiers du fils gauche et est inférieur à tous les entiers du fils droit. On utilisera le même type que dans la partie "Arbres binaires".

1. Définissez une fonction `estabr` prenant en argument un arbre binaire et indiquant si c'est un arbre binaire de recherche.
2. Définissez une fonction `quicksearch` similaire à `appartient` mais travaillant sur des arbres binaires de recherche et étant récursive terminale.
3. Définissez une fonction `inser` prenant en arguments un arbre binaire de recherche et un entier et modifiant l'arbre pour y insérer l'entier de telle sorte que l'arbre obtenu reste un arbre binaire de recherche.
4. Définissez une fonction `suppr` prenant en arguments un arbre binaire de recherche et un entier et modifiant l'arbre pour y supprimer l'entier de telle sorte que l'arbre obtenu reste un arbre binaire de recherche.
5. Définissez une fonction `changeroot` prenant en arguments un arbre binaire de recherche et un entier appartenant à l'arbre et modifiant l'arbre de telle sorte qu'il reste un arbre binaire de recherche, contienne toujours les mêmes données, et que l'entier donne en argument devienne la racine.
6. On dit qu'un arbre binaire de recherche est équilibré lorsque pour chacun de ses noeuds, la hauteur du fils gauche et du fils droit diffèrent au plus de un. Définissez une fonction prenant en argument un arbre binaire de recherche et indiquant s'il est équilibré.
7. (difficile) Définissez une fonction `equilibre` prenant en arguments un arbre binaire de recherche et renvoyant un arbre binaire de recherche équilibré représentant les mêmes données.

9 Graphes

Un graphe est un objet mathématique qui représente des points (appelés "sommets") reliés entre eux (un lien entre deux sommets est appelé "arête"). Les sommets de nos graphes seront numérotés par des entiers. Une arête sera donc représentée par deux entiers (le sommet de départ et celui d'arrivée). Nos graphes seront non orientés, c'est à dire qu'une arête qui va du sommet 1 au sommet 2 est la même chose qu'une arête qui va du sommet 2 au sommet 1.

1. Définissez un type `sommet`, un type `arete` et un type `graphe`.
2. Avec le type `graphe` qu'on vient de définir, il est possible d'avoir des arêtes faisant référence à des sommets inexistants. Écrivez une fonction `check_graphe` prenant en argument un graphe et vérifiant que ce n'est pas le cas.
3. Définissez une fonction `adjacents` qui prend en arguments un graphe et un sommet de ce graphe et renvoie la liste des sommets adjacents au sommet donné en argument (c'est à dire reliés à ce sommet par une arête).
4. (difficile) Définissez une fonction `connexe` qui prend en argument un graphe et renvoie la liste de ses composantes connexes.

10 Calcul formel

On s'intéresse à des expressions composées de nombres réels (type `float`), variables (nommées par un unique caractère de type `char`), et opérations classiques (addition, opposé, multiplication, inverse, exponentielle, logarithme népérien).

- Écrivez un type `expr`.
- Représentez l'expression $5.3 * X + 3/A - 2$
- Écrivez une fonction `soustraction` prenant en arguments deux expressions A et B et renvoyant une expression équivalente à $A - B$.
- Écrivez une fonction `division` prenant en arguments deux expressions A et B et renvoyant une expression équivalente à A/B.
- Écrivez une fonction `puissance` prenant en arguments deux expressions A et B et renvoyant une expression équivalente à A^B.
- Écrivez une fonction `derive` prenant en arguments une variable (*ie* un caractère) et une expression, renvoyant la dérivée de l'expression selon la variable donnée.
- On veut maintenant évaluer nos expressions. Il faut pour cela donner une valeur à chaque variable. On appelle contexte une structure permettant d'associer des valeurs à des variables. Définissez un type `contexte` adapté.
- Écrivez une fonction `evaluation` prenant en arguments une expression et un contexte et renvoyant l'évaluation de l'expression en prenant pour les variables les valeurs précisées dans le contexte.
- Évaluez l'exemple plus haut avec $X = -2$ et $A = 41.7$.