TD 6 - Projets

Projet 1. Arbres binaires

On cherche à créer un type arbre binaire : un arbre binaire est soit un **arbre vide**, soit un **noeud** *racine* étiqueté (dans notre cas par un entier) et renvoyant vers deux sous-arbres, l'un étant appelé le **sous-arbre gauche** (SAG), l'autre étant appelé le **sous-arbre droit** (SAD). Chacun de ces sous-arbres est à son tour un arbre binaire. Si ces deux sous-arbres *fils* sont vides, l'arbre *père* est une **feuille**. Un arbre binaire est dit **saturé** quand deux *frères* (*fils* d'un même *père*) y sont toujours soit tous les deux vides soient tous les deux non vides.

1. Pour ces arbres binaires, créer un type `arbrebinaire` qui correspond soit à un arbre `Vide` soit à un `Noeud`.

2. Créer les arbres binaires suivants : un arbre `v` vide, un arbre binaire `f` qui est une feuille étiquetée par 1 ; un arbre binaire `a` qui est étiqueté par 25 et dont les deux sous-arbres sont des feuilles étiquetées par 1 ; un arbre binaire `s` qui est étiqueté par (-3) et dont le sous-arbre gauche est vide et le sous-arbre droit est une feuille étiqueté par 1.

3. Parmi les arbres ci-dessus, lesquels sont saturés ?

4. A un type structuré on associe généralement des fonctions de base (appelées *primitives*) qui permettent de créer d'autres fonctions plus sophistiquées sur ce type. Créer les primitives suivantes.

- `estvide` qui teste si un arbre est vide.
- `estfeuille` qui teste si un arbre est une feuille.
- `etiquette` qui donne l'étiquette d'un arbre non vide.
- `sag` (resp. `sad`) qui retourne le SAG (resp. SAD) d'un arbre non vide.
- `estsature` qui indique si un arbre a un type conforme au type saturé.

5. En utilisant les primitives ci-dessus autant que de besoin, créer les fonctions suivantes.

- `taille` qui retourne le nombre noeuds d'un arbre.
- `hauteur` qui retourne le nombre maximal de noeuds entre la racine et une feuille (par conséquent : 0 pour un arbre vide, 1 pour un noeud racine isolé).
- Fonction qui retourne la liste des étiquettes d'un arbre ; réaliser trois variantes de cette fonction :
  – `prefixe` : l'étiquette d'un noeud apparait **avant** les étiquettes des arbres fils.
  – `postfixe` : l'étiquette d'un noeud apparait **après** les étiquettes des arbres fils.
  – `infixe` : l'étiquette d'un noeud apparait après celles du SAG et avant celles du SAD.
- `feuillage` qui retourne la liste des étiquettes des feuilles d'un arbre.
- `lister` qui retourne une chaîne de caractères représentant un arbre sous forme de listes emboitées. Avec les exemples précédents, on aura : lister v = `"[]"`, lister f = `"[1;[];[]]"`, lister a = `"[25;[1;[];[]];[1;[];[]]]"`, lister s =`"[-3;[];[1;[];[]]]"`.
- `arborer` qui crée un arbre binaire à partir d'une chaine représentant un arbre, en utilisant des fonctions annexes selon besoins. On contrôlera à mesure que la configuration de la chaine permette bien de construire un arbre.
  En Caml, peut-on utiliser pour cette fonction une liste emboitée au lieu d'une chaîne de caractères ? (par exemple `[1;[];[]]` au lieu de `"[1;[];[]]"`).

Projet 2. Listes d'entiers

Définir, sur le modèle du type arbre binaire, un type `listedentier`. Le but est de réaliser avec ce type tout ce que l'on pourrait faire en utilisant un type `int list` et les opérateurs et fonctions de référence sur les listes (hd, tl, ::, @).

Projet 3. ABR

Un arbre binaire de recherche (ABR) est un arbre binaire (pas forcément saturé) qui est étiqueté par des valeurs totalement ordonnables (dans notre cas des entiers) grâce à deux règles de construction :

L'étiquette d'un noeud est

(1) supérieure ou égale à celle de chaque noeud de son SAG, et

(2) strictement inférieure à celles de chaque noeud de son SAD.

1. Quelles seront les principales différences entre ce type et le précédent ? Donner un exemple d'arbre binaire étiqueté respectant la propriété (1) mais pas la propriété (2). Donner deux exemples d'ABR dont au moins un non trivial.

2. On définit le type ABR par : `type abr=arbrebinaire` ; on disposera ainsi de toutes les fonctions définies pour les arbres binaires. Créer une fonction estABR qui vérifie si un arbre de type `abr` respecte la définition formelle.

3. Ecrire les fonctions qui :

- Ajoute une valeur dans un arbre de recherche.
- Supprime une valeur passée en paramètre dans un arbre de recherche.
- Retourne la liste des valeurs contenues dans un arbre binaire de recherche, par ordre croissant.
- Retourne la liste des valeurs contenues dans un arbre binaire de recherche, par ordre décroissant.
- Construit un arbre binaire de recherche à partir d'une liste. Est-ce important que la liste soit préalablement triée ?
- ...

Projet 4. Le monnayeur

A réaliser par groupe de quatre en répartissant les tâches.

On cherche à modéliser le monnayeur d'une machine à café.

La machine accepte en entrée toutes les pièces de 10 ct à 2 eu, déduit la somme correspondant à la commande et rend la monnaie. Pour ce faire, elle dispose d'une réserve de pièces qui est alimentée par les pièces fournies par les usagers ou par une recharge effectuée par l'exploitant. L'exploitant possède un mot de passe qui lui permet d'effectuer toutes les opérations de maintenance (retraits ou ajouts de pièces).

Si la machine trouve le montant exact ou si elle peut rendre la monnaie, elle donne la boisson, et rend l'éventuelle monnaie, sinon elle refuse les pièces en bloc.

1. Définir les constantes : prix d'une boisson (prix unique : 40 ct), mot de passe de l'exploitant, etc.

2. Choisir les types des différentes variables : réserve de pièces de la machine, commande d'une boisson, rendu de monnaie, retrait ou recharge en pièces, etc.

3. Définir les fonctions nécessaires : vérification d'une commande, vérification du rendu, rendu effectif de la monnaie et "fourniture" de la boisson, rejet d'une commande, passage en mode maintenance, mise à jour de la réserve de pièces (ajout ou retrait), etc.

4. Ecrire le programme qui fait tourner la machine. Il n'est pas interdit de l'améliorer : messages à l'utilisateur (`print_string`), modification du prix des boissons, réductions promotionnels, commandes multiples, prix différents selon la commande, etc.