DS2 d'informatique MPSI
16 mars 2012

Devoir surveillé

Durée : 2 heures

Aucun document n'est autorisé. **On pourra définir des fonctions non demandées explicitement, si cela facilite la programmation. Inutile de prouver la correction et la terminaison des fonctions écrites, sauf si on le demande explicitement. Les fautes de syntaxe seront sanctionnées.**

Représentations d'ensembles avec des intervalles

Dans tout le problème, les boucles impératives for et while sont interdites, ainsi que les références.

Les fonctions Caml min max fst snd sont autorisées.

De nombreux algorithmes reposent sur la manipulation d'ensembles d'éléments ordonnés. Lorsque ces ensembles contiennent de nombreux éléments adjacents (*i.e.* aucune valeur n'existe entre ces deux éléments), il est plus performant en terme de temps de calcul et d'occupation mémoire de manipuler des intervalles au lieu de valeurs singulières. Cela permet également de manipuler des ensembles infinis sous la forme d'un ensemble fini d'intervalles contenant un nombre de valeurs infinies (*i.e.* dans $\mathbb{Q}$ ou $\mathbb{R}$).

L'objectif de ce problème est de comparer deux implantations différentes d'un ensemble d'entiers, la première à base de listes triées d'intervalles, et la seconde à base d'arbres binaires d'intervalles.

- Nous nous limiterons à des intervalles fermés de $\mathbb{R}$ dont les bornes sont des entiers $[n_{\min}, n_{\max}]$ (l'extension à des intervalles ouverts et aux valeurs $-\infty$ et $+\infty$ ne pose par de problème majeur).
- Nous ferons l'hypothèse que les couples qui représentent des intervalles sont bien formés, *i.e.* la valeur représentant le minimum est inférieure ou égale à la valeur représentant le maximum.
- **Intervalles disjoints :** deux intervalles sont dits *disjoints* si leur intersection est vide.
- **Fusion de deux intervalles :** la *fusion* de deux intervalles a comme minimum le plus petit des minima des deux intervalles, et comme maximum le plus grand des maxima des deux intervalles. Cette opération correspond à l'union des intervalles lorsque ceux-ci ne sont pas disjoints.

Un intervalle est représenté par le type intervalle, équivalent à une paire de int (son minimum en premier et son maximum en second) :

```
# type intervalle == int * int;;
Type intervalle defined.

# let (i0 : intervalle) = 3, 6;;
i0 : intervalle = 3, 6

# fst i0;;
- : int = 3
```

Partie A – Fonctions générales

A.1 Écrire une fonction disjoints de type intervalle -> intervalle -> bool testant si deux intervalles sont disjoints.

A.2 Écrire une fonction fusion de type intervalle -> intervalle -> intervalle fusionnant les intervalles pris en arguments.

Partie B – Représentation par des listes triées d'intervalles

La réalisation la plus simple d'un ensemble de valeurs en utilisant des intervalles consiste à utiliser une liste (chaînée) d'intervalles. Nous utiliserons plus précisément une liste *triée* d'intervalles entiers.

Liste bien formée d'intervalles : une *liste bien formée d'intervalles* est une liste d'intervalles qui respecte les contraintes suivantes :

1. les intervalles sont bien formés.
2. les intervalles sont disjoints deux à deux.
3. la liste est triée selon la relation d'ordre suivante : un intervalle i_1 est strictement plus petit qu'un intervalle i_2 si et seulement si le maximum de i_1 est strictement plus petit que le mimimum de i_2.

Clairement, une liste triée d'intervalles bien formés (selon cette relation) est une liste bien formée d'intervalles (inutile de le montrer).

Une liste triée d'intervalles est représentée par le type `liste` équivalent à une liste d'intervalles :

```
# type liste == intervalle list;;
Type liste defined.
```

B.1 Écrire une fonction ajouter de type `intervalle -> liste -> liste` telle que `ajouter i l` renvoie la liste bien formée d'intervalles contenant les intervalles contenus dans la liste `l` qui sont disjoints de `i`, et :

1. soit le résultat de la fusion de `i` et des intervalles contenus dans `l` qui ne sont pas disjoints de `i`.
2. soit l'intervalle `i`, si tous les intervalles contenus dans la liste `l` lui sont disjoints.

B.2 Donner des exemples de valeurs des arguments `i` et `l` de la fonction ajouter qui correspondent au meilleur et au pire des cas en nombre d'appels récursifs effectués, en précisant ce nombre d'appels.

B.3 Écrire une fonction appartenir : `int -> liste -> bool` testant si un entier appartient à une liste bien formée d'intervalles.

B.4 Écrire une fonction `verifier : liste -> bool` testant si une liste d'intervalles (bien formés) est bien formée.

Partie C – Représentation par des arbres binaires

L'utilisation de la structure d'arbre binaire pour représenter un ensemble à base d'intervalles permet de réduire la complexité en temps de calcul pour les différentes opérations.

Un ensemble à base d'intervalles est ici représenté par un arbre binaire dont les feuilles sont les intervalles :

- **Arbre binaire d'intervalles :** un *arbre (binaire) d'intervalles* a est une structure qui peut :
  1. soit être vide (notée $\emptyset$).
  2. soit être une feuille étiquetée par un intervalle (on dira aussi que la feuille et a contiennent l'intervalle).
  3. soit un nœud qui contient un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'intervalles non vides.
- L'ensemble des feuilles (resp. des nœuds) d'un arbre a est noté $\mathcal{F}(a)$ (resp. $\mathcal{N}(a)$).
- **Intervalle englobant :** l'*intervalle englobant* d'un arbre d'intervalles est le plus petit intervalle bien formé qui contient tous les intervalles contenus dans les feuilles de l'arbre. L'intervalle englobant d'un arbre d'intervalles a est noté $\mathcal{I}(a)$.
- **Parcours gauche-droite :** le parcours gauche-droite d'un arbre binaire non vide a, noté $\mathcal{E}(a)$, construit la séquence des feuilles de l'arbre selon un certain ordre. Cette séquence est définie comme étant la liste d'unique terme l'étiquette de a si a est une feuille, et la concaténée de $\mathcal{E}(\mathcal{G}(a))$ et de $\mathcal{E}(\mathcal{D}(a))$ sinon.
- **Arbre bien formé d'intervalles :** un *arbre bien formé d'intervalles* est un arbre d'intervalles tel que :
  1. les intervalles contenus dans les feuilles sont bien formés.
  2. les intervalles contenus dans les feuilles sont disjoints deux à deux.
  3. la liste d'intervalles obtenue par le parcours gauche-droite de l'arbre est bien formée.

C.1 Soit a un arbre d'intervalles bien formés. Montrer que si pour tous les nœuds de l'arbre, le maximum de l'intervalle englobant du fils gauche est strictement plus petit que le minimum de l'intervalle englobant du fils droit, alors l'arbre est bien formé.

Un arbre binaire d'intervalles est réprésenté par le type Caml :

```
# type arbre =
        | Vide
        | Feuille of intervalle
        | Noeud of arbre * arbre;;

Type arbre defined.
```

Remarque : on notera que le type arbre ainsi crée contient par exemple Noeud (Vide, Feuille (3, 4)), ce que notre définition d'arbre binaire d'intervalles interdit.

C.2 Écrire une fonction englobant de type arbre -> intervalle, associant à un arbre non vide l'intervalle l'englobant.

C.3 Écrire une fonction verifier : arbre -> bool testant si un arbre d'intervalles (bien formés) est bien formé.

C.4 Écrire une fonction appartenir : int -> arbre -> bool testant si l'entier pris en argument appartient à un intervalle contenu dans l'arbre pris en argument.

On introduit une fonction fusion_arbre, que l'on pourra librement utiliser dans la suite :

```
# let fusion_arbre = fun
        | Vide d -> d
        | g Vide -> g
        | g d -> Noeud (g, d);;
fusion_arbre : arbre -> arbre -> arbre = <fun>
```

qui, lorsqu'elle prend en arguments des arbres bien formés d'intervalles g et d, *tels que le maximum de l'englobant de g soit strictement inférieur au minimum de l'englobant de d*, associe un arbre bien formé d'intervalles.

C.5 Écrire une fonction decouper : int -> arbre -> arbre * arbre * arbre telle que l'appel decouper v a renvoie le triplet composé successivement :

1. d'un arbre bien formé contenant les intervalles (contenus dans a) strictement inférieurs à la valeur v.
2. d'un arbre bien formé contenant les intervalles (contenus dans a) contenant la valeur v.
3. d'un arbre bien formé contenant les intervalles (contenus dans a) strictement supérieurs à la valeur de v.

C.6 Écrire une fonction ajouter : intervalle -> arbre -> arbre telle que l'appel ajouter i a renvoie un arbre bien formé d'intervalles contenant les intervalles contenus dans a disjoints de i, et :

1. soit le résultat de la fusion de i et des intervalles contenus dans a qui ne sont pas disjoints de i
2. soit i, si i est disjoint de tout intervalle contenu dans a.

Partie D – D'une représentation à l'autre

D.1 Proposer une fonction de conversion d'une liste bien formée d'intervalles en un arbre bien formé des mêmes intervalles.

D.2 Proposer une fonction de conversion d'un arbre bien formé d'intervalles en une liste bien formée des mêmes intervalles.