ENSIIE 2012–2013 Programmation fonctionnelle

Examen 21/01/2013 — 1h45

Tout appareil électronique est interdit et doit être rangé dans un sac fermé. Les notes de cours sur papier sont autorisées. Lisez tout l'énoncé (3 pages) avant de commencer, lisez attentivement les questions, vérifiez que les réponses que vous proposez correspondent aux questions posées ! Ça va mieux en le disant...

Il est fortement recommandé de commencer par l'exercice 1.

Exercice 1 [Entiers en binaire]

On souhaite représenter des entiers codés en binaire à l'aide du type `chiffre` défini par les deux constructeurs constants `Zero` et `Un`. Un entier est dès lors décrit par une liste d'éléments de type `chiffre` se lisant de droite à gauche, l'élément le plus à gauche étant le chiffre de poids le plus faible (celui des unités).

Par exemple, l'entier 14 dont l'écriture en binaire est 1110 sera représenté par la liste : `Zero::Un::Un::Un::[]`.

Questions

1. Proposer une fonction qui, sur la donnée d'une représentation en binaire de l'entier n, retourne la représentation en binaire de l'entier $n + 1$. Préciser en commentaires le type de cette fonction.
2. Proposer une fonction qui, sur la donnée d'une liste d'éléments de type `chiffre`, retourne la valeur entière représentée.
3. Démontrer proprement que, pour tout entier x, la fonction h suivante vérifie $h\ x\ l = x^n$ où n est l'entier représenté par la liste l.

```
let rec h x l = match l with
| [] -> 1
| Zero::r -> h (x * x) r
| Un::r -> x * (h (x * x) r)
```

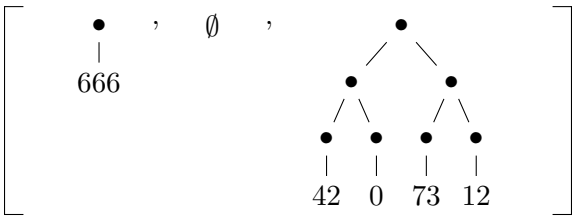
Exercice 2 [une variante de liste]

On souhaite définir une structure de données permettant de représenter des *séquences* c'est-à-dire des ensembles dont l'ordre d'entrée des éléments est conservé. On pourrait penser à prendre des listes mais les listes ne permettent pas un accès efficace à un élément désigné par son indice dans la séquence (l'indice du dernier élément entré étant toujours 1).

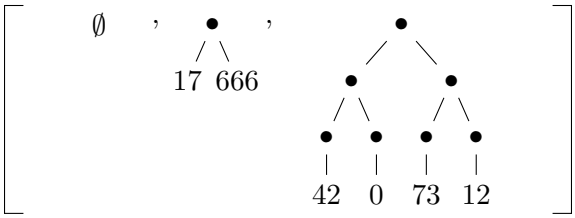
On se propose de réaliser une telle structure en s'appuyant sur la représentation en binaire du nombre d'éléments. À chaque position i, dans la représentation binaire du nombre d'élément, si le bit est à 1, notre structure contiendra un arbre d'éléments qui contient **exactement** 2^i éléments, sinon elle contiendra un arbre vide. On peut donc envisager de représenter les séquences comme des listes d'arbres, un peu comme dans l'exercice précédent, avec le bit de poids faible à gauche. Attention à ne pas confondre les éléments de la séquence et les arbres !

On considérera des arbres ne contenant des valeurs que dans les feuilles.

Voilà par exemple la séquence contenant 5 éléments (écriture binaire 101), les éléments étant 12, 73, 0, 42, 666 rentrés *dans cet ordre*. L'élément d'indice 1 est 666 (le dernier rentré) ; l'élément d'indice 4 est 73.



Si on lui ajoute un sixième élément (écriture binaire de 6 : 110), disons 17, on obtient la séquence suivante :



Notons que, maintenant, l'indice de l'élément 666 est 2, l'élément d'indice 1 étant 17.

Questions

1. Proposer un type pour les arbres d'éléments.
2. Proposer un type pour les séquences. Déclarer une valeur `vide` représentant une séquence ne contenant aucun élément.
3. Proposer une fonction `cardinal` qui, sur la donnée d'une séquence, retourne le nombre d'éléments qu'elle contient. Rappel : la séquence est une liste d'arbre d'éléments et le i^{e} arbre de cette liste est :
   – Vide si le i^{e} bit du nombre d'éléments est 0,
   – Un arbre binaire complet (c'est-à-dire dont toutes les branches sont de la même longueur) contenant *exactement* 2^i éléments stockés dans les 2^i feuilles sinon.
4. Proposer une fonction `applique` qui, sur la donnée d'une fonction f et d'une séquence s d'éléments s_i retourne la séquence des $(f\ s_i)$ (dans cet ordre). Par exemple si s est la séquence contenant les éléments 1, 666 et 42 dans cet ordre, `applique` $f\ s$ retournera la séquence contenant $(f\ 1)$, $(f\ 666)$ et $(f\ 42)$ dans cet ordre.
5. Proposer une fonction `dans_arbre` qui, sur la donnée d'un arbre t complet, d'un entier k (forcément une puissance de 2) tel que t contient exactement k éléments et d'un indice i, retourne le i^{e} élément de t à partir de la gauche, c'est-à-dire la valeur contenue dans la i^{e} feuille de t. Cette fonction lèvera l'exception `Not_found` si l'arbre ne contient pas de i^{e} élément.
6. Combien d'indices dans la séquence passe-t-on entre le i^{e} et le $(i+1)^{\text{e}}$ arbre :
   – Si le i^{e} arbre n'est pas vide ?
   – Si le i^{e} arbre est vide ?
7. Proposer une fonction `ieme` qui, sur la donnée d'une séquence, retourne la valeur d'indice i. Cette fonction lèvera l'exception `Not_found` si la séquence ne contient pas de i^{e} élément.

8. Estimer le nombre d'opérations nécessaire pour obtenir le i^{e} élément d'une séquence. Qu'en serait-il si on avait choisi une liste plutôt qu'une séquence ?
9. Proposer une fonction `cons` qui, sur la donnée d'une séquence l et d'un élément e retourne une séquence dont le premier élément (à l'indice 1) est e et dont les suivants sont les éléments de l (dans le même ordre). On veillera à conserver la propriété : le i^{e} arbre de la séquence résultat est :
   – Vide si le i^{e} bit du nombre d'éléments est 0,
   – Un arbre binaire complet contenant *exactement* 2^i éléments stockés dans les 2^i feuilles sinon.