Série d'exercices #2

IFT-2035

May 4, 2021

2.1 Filtrer des éléments

Compléter la fonction *less* qui filtre les éléments d'une liste pour n'en garder que les éléments plus petits qu'une valeur seuil[a]:

$$\begin{array}{l} \mathit{less} \ :: \mathsf{Int} \to [\mathsf{Int}] \to [\mathsf{Int}] \\ \mathit{less}\ k\ \ldots = \ldots \end{array}$$

$$\mathit{less}\ 7\ [1,7,2,5,9,3,10] \quad \rightsquigarrow^* \quad [1,2,5,3]$$

Dans chacun des cas suivants, décrire la valeur retournée par l'expression et expliquer comment elle est calculée (i.e. montrer les étapes de l'évaluation).

1. *less* 5 []
2. *less* 5 [4, 5, 6]
3. *less* 5 [7, 3, 6, 2]

[a] La notation $e \rightsquigarrow^* e'$ s'utilise pour indique que l'expression e peut être réduite à e' en un nombre non-spécifié de β-réductions

2.2 Conversion de base

Écrire les fonctions suivantes en Haskell pour convertir des nombres en représentation binaire à décimal et vice versa (à remarquer que l'argument est un entier et que par exemple l'entier "cent-un" représente le nombre binaire "un-zéro-un" qui correspond à 5 en décimal). Vous aurez besoin des fonctions prédéfinies mod, et div.

$$\textit{bin2dec}\ 10001 \quad \rightsquigarrow^* \quad 17$$
$$\textit{dec2bin}\ 17 \quad \rightsquigarrow^* \quad 10001$$

Écrire la fonction *baseconv* en Haskell qui convertit d'une base à une autre (≤ 10). N'hésitez pas à définir des fonctions auxiliaires si nécessaire.

$$\textit{baseconv}\ 2\ 10\ 10001 \quad \rightsquigarrow^* \quad 17$$
$$\textit{baseconv}\ 10\ 2\ 17 \quad \rightsquigarrow^* \quad 10001$$

Donner aussi le type de chacune des fonctions que vous avez définies.
Note: La distinction entre une valeur et sa représentation est un thème qui réapparait souvent dans ce cours, par exemple sous la forme de la différence entre la syntaxe et la sémantique des programmes.

2.3 Quicksort

Implanter en Haskell une variante de *quicksort* pour des listes d'entiers. En clair, trier une liste comme suit:

1. choisir un élément, que l'on nommera le pivot.
2. partitionner la liste en deux sous-listes d'éléments plus petits resp. plus grands que le pivot.
3. trier les deux sous-listes.
4. combiner ces sous-listes triées et le pivot en une liste triée.

Le type sera: $\textit{quicksort} :: [\mathsf{Int}] \to [\mathsf{Int}]$.
Il faudra peut-être définir une ou plusieurs fonctions auxiliaires.
L'opération de concaténation de deux listes s'écrit ++ en Haskell:

$$[1,2] ++ [4,5,6] \quad \equiv \quad (++)\ [1,2]\ [4,5,6] \quad \rightsquigarrow^* \quad [1,2,4,5,6]$$

Finalement, généraliser la fonction de tri précédente pour pouvoir l'appliquer à des listes quelconques (pas seulement [Int]), en passant un argument supplémentaire qui indique l'opération de comparaison à utiliser.
Donner aussi le type de cette fonction plus générale et de toutes les fonctions auxiliaires que vous avez définies.

2.4 Micro optimizer

Soit le code Haskell ci-dessous:

```
data Exp = Enum Int           -- Une constante
         | Eplus Exp Exp      -- e1 + e2
         | Etimes Exp Exp     -- e1 * e2

optimize :: Exp -> Exp
```

Exp est un type qui représente des expressions simples incluant uniquement des opérateurs arithmétiques. Écrire la fonction *optimize* qui va essayer de simplifier une expression en éliminant *toutes* les multiplications par 1 et 0, ainsi que les additions à 0.