TP6 Programmation fonctionnelle

L3 Informatique Nancy

14 mars 2022

1 Exceptions et types *result*

Exercice 1 *1. Définissez le type inductif int_tree des arbres binaires dont les feuilles sont des entiers et dont les nœuds sont sans étiquette.*

2. *Écrivez naïvement une fonction tree_mem : int -> int_tree -> bool qui prend en argument un entier n et un arbre t et renvoie* ***true*** *si n apparaît dans t et* ***false*** *sinon.*
3. *Réécrivez tree_mem (même spécification) en utilisant cette fois une fonction auxiliaire qui lance une exception quand une occurrence est trouvée dans l'arbre.*
4. *Créez un arbre binaire complet de très grande taille dont toutes les feuilles sont* 0*. Comparez le temps de calcul*[1] *de tree_mem 0 sur cet arbre pour les deux implémentations. Pourquoi la deuxième implémentation est optimisée par rapport à la première ?*

Exercice 2 *1. On rappelle qu'OCaml fournit un mécanisme fonctionnel simple qu'on peut substituer aux exceptions : le type result qui est défini comme suit :*

```
type ('a, 'e) result = Ok of 'a | Error of 'e
```

Traduisez précisément votre implémentation de tree_mem de la question 1.3 pour remplacer les exceptions par des types result.

2. *Testez le temps de calcul de tree_mem 0 sur l'arbre de la question 1.4 pour cette nouvelle implémentation.*

Exercice 3 *Réimplémentez l'opérateur Result.bind : ('a, 'e) result -> ('a -> ('b, 'e) result) -> ('b, 'e) result que vous avez vu en cours. (Pour bien comprendre la sémantique de cet opérateur, n'hésitez pas à revoir les exemples du cours.)*

2 Continuations

Exercice 4 *On considère fact la fonction qui renvoie la factorielle d'un entier naturel. Traduisez-la en une fonction fact_cps écrite en style par passage de continuation.*

Exercice 5 *On considère depth_tree qui renvoie la hauteur d'un int_tree. Traduisez-la en une fonction depth_tree écrite en style par passage de continuation.*

Exercice 6 *1. On reprend l'implémentation de tree_mem de la question 1.3, mais cette fois on va modéliser les exceptions par des continuations. Pour cela, comme dans le cours, utilisez une fonction auxiliaire (analogue à la fonction auxiliaire qui pouvait lancer une exception) avec deux continuations, l'une à appeler quand le calcul va jusqu'au bout, l'autre à appeler quand une occurrence est trouvée dans l'arbre.*

2. *Testez le temps de calcul de tree_mem 0 sur l'arbre de la question 1.4 pour cette nouvelle implémentation.*

1. On pourra utiliser Sys.time() qui donne le temps processeur utilisé depuis le lancement de l'instance OCaml.