EPITA ING1 2010 S2 PROGRAMMATION FONCTIONNELLE

Didier Verna

Documents et calculatrice interdits
Toute réponse non justifiée sera comptée comme nulle
Ce partiel il est trop fastoche

1 Ordre Supérieur *(2 points)*

Un langage de programmation est dit *fonctionnel* lorsque les fonctions sont des objets d'« ordre supérieur ». La notion d'ordre supérieur (ou de « premier ordre », ou encore de « première classe ») est due à Christopher Strachey. Citez 4 aspects caractérisant de telles fonctions.

2 Techniques d'Évaluation *(6 points)*

1. Nommez et décrivez les deux grands principes d'évaluation sur lesquels se basent Lisp d'un côté, Haskell de l'autre.
2. Considérons la fonction Haskell suivante :

   ```
   ifnot :: Bool -> a -> a -> a
   ifnot test e1 e2 = if test then e2 else e1
   ```

   Décrivez précisément l'évaluation de l'expression suivante :

   ```
   ifnot False "OK" (error "Unexpected test outcome")
   ```

3. Considérons la fonction Lisp suivante :

   ```
   (defun ifnot (test e1 e2)
     (if test e2 e1))
   ```

   Décrivez précisément l'évaluation de l'expression suivante :

   ```
   (ifnot nil "OK" (error "Unexpected test outcome"))
   ```

   Pourquoi cette fonction est-elle problématique ? Quelle est la solution adoptée par Lisp pour les fonctions *built-in* de ce type ?
4. Sauriez-vous décrire une ou plusieurs techniques lispiennes permettant aux utilisateurs de s'écrire leur propre `ifnot` correctement ?

3 Arguments Fonctionnels *(6 points)*

Le « folding » (ou « réduction ») est une abstraction importante disponible grâce aux fonctions du premier ordre. En Haskell, la fonction de folding simple est la suivante (l'équivalent Lisp est `reduce`) :

```
foldr1 :: (a -> a -> a) -> [a] -> a
```

1. Décrivez cette fonction (ses arguments, ce qu'elle fait, ce qu'elle renvoie *etc.*), et donnez un exemple.
2. Expliquez son nom.
3. La fonction de folding généralisée se nomme `foldr` en Haskell (en Lisp, c'est toujours `reduce` qui est utilisée). Qu'est-ce que le folding généralisé par rapport au folding simple ?
4. Donner le type Haskell de `foldr` et expliquez-le.

4 Preuve de Programme *(6 points)*

Soit la fonction Haskell suivante :

```
s :: [Int] -> Int
s [] = 0
s (x:xs) = x + s xs
```

1. Quel est le but de cette fonction ?
2. Prouvez par induction que $\forall xs, ys$, `sum (xs ++ ys) = sum xs + sum ys`
   Remarques :
   - Toutes les propriétés souhaitées de `(++)` seront considérées acquises.
   - Commencez par l'induction sur `x`.
3. Étendez votre preuve aux *fp-listes* (listes partiellement définies).