EPITA ING1 2012 S2 PARTIEL PFON

Didier Verna

Documents et calculatrice interdits

Toute réponse non justifiée sera comptée comme nulle.
Il est préférable de ne rien répondre que de tenter d'inventer d'importe quoi. Toute tentative de bluff sera sanctionnée par un malus.

1 Pureté *(3 points)*

1. Parmi les langages fonctionnels, certains sont dits « purs ». Expliquez ce que cela signifie.
2. Citez deux avantages de la pureté par rapport aux langages fonctionnels impurs.

2 Typage Statique et Dynamique *(4 points)*

Considérons la fonction Haskell suivante.

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

1. Que fait cette fonction ?
2. Que donne l'évaluation de `bad ['a', 'b', 'c']` ?
3. Quel est le problème, et quelle solution y apporteriez-vous ?
4. A-t-on ce problème en Lisp (justifiez) ?

3 Techniques d'Évaluation *(5 points)*

Considérons le code Haskell suivant.

```
foo :: Float
foo = 2 * foo

bar :: Float -> Float -> Float
bar x 0 = 0
bar 0 y = 0
bar x y = x * y
```

1. Expliquez l'évaluation de `bar foo 0`. Quel est le résultat ?
2. Expliquez l'évaluation de `bar 0 foo`. Quel est le résultat ?
3. Notez que la fonction `bar` est écrite de manière équationnelle. Réécrivez cette fonction en utilisant cette fois des guardes (et en respectant l'ordre des équations).
4. En utilisant maintenant cette nouvelle définition, répondez à nouveaux aux deux premières questions.

4 Formes de scoping *(3 points)*

Considérons la fonction Lisp suivante :

```
(defun foo (n)
  (lambda (x) (+ n x)))
```

1. Rappelez les définitions du scoping lexical et dymanique.
2. Rappelez la définition d'une fermeture lexicale.
3. Que fait cette fonction ?
4. Que vaut le résultat de l'expression suivante :

```
(let ((n 10))
  (funcall (foo 5) 1))
```

5 Fonctions de Première Classe *(5 points)*

Soit la fonction `multiply` définie comme prenant deux flottants en argument et renvoyant leur produit. Cette fonction s'écrit simplement `multiply a b = a * b` en Haskell.

1. Complétez cette définition en fournissant son prototype (afin de renseigner Haskell sur le type de cette fonction).
2. Que représente l'expression `(multiply 2)` et comment appelle-t-on cette facilité ? Quel est son type ?
3. Quelle est l'associativité de l'opérateur de typage en Haskell ? Même question pour l'application de fonction.
4. Que représente l'expression `(*2)` et comment appelle t'on cette facilité ?
5. Écrivez l'équivalent de `(*2)` en Lisp.