Examen

Session 2

15 juin 2017

Durée 3h. Tous documents papier autorisés. Appareils électroniques et communication de toute forme interdits. Veillez à éteindre et ranger vos téléphones portables dans vos sacs pendant toute la durée de l'examen. Veuillez lire le sujet dans son intégralité avant de commencer.

1 Exercices

1. Donner le type et la valeur des expressions suivantes :
   (a) `2+2`
   (b) `List(1,2,3)`
   (c) `List(1, "deux", true)`
   (d) `(2+2).toString+2`
   (e) `{val x=3; x*x}+1`
   (f) `(x:Int) => x+1`
   (g) `{val f=(x:Int) => x+1; f(3)}`
   (h) `print("Hello")`
   (i) `Some(42)`
   (j) `Some(Some(None))`
2. Quelle est la valeur de l'expression suivante ?

```
{def inc(x:Int) = x+1
 def twice(f:Int => Int) = (x:Int) => f(f(x))
 val f = twice(inc); f(0)}
```

3. Donner la fonction de signature :

```
def assoc[A,B,C] (x: ((A, B), C)): (A, (B, C))
```

4. Donner la fonction de signature :

```
def curry2[A,B,C,D] (f: A=>B=>C=>D): (A,B,C) => D
```

5. On trouve dans la classe prédéfinie `List[A]` une méthode :

   `def lastIndexWhere(p: (A) => Boolean, end: Int): Int`

   Finds index of last element satisfying some predicate before or at given end index.

   `p` *the predicate used to test elements.*

   returns *the index ≤ end of the last element of this sequence that satisfies the predicate p, or -1, if none exists.*

   Utilisez celle-ci pour définir une fonction :

`def pivot(l: List[Int], n:Int): Int`

retournant l'index dans la liste strictement croissante d'entiers `l` du plus grand élément inférieur à `n` (exemple : `pivot(List(1,3,5,7,9), 6) = 2`). Attention au cas où le plus grand élément est le dernier (exemple : `pivot(List(1,3,5,7,9), 15) = 4`)!

6. Définir la fonction récursive :

   `def exists[A](l: List[A], p: A=>Boolean): Boolean`

   qui retourne `true` si `p(x)` vaut `true` pour au moins un élément `x` de la liste `l`.

7. Redéfinir `exists` à l'aide de la méthode de parcours des listes :

   `def foldLeft[B](z: B)(op: (B, A) => B): B`

2 Problème

Un des problèmes que posent les chaîne de caractère dans la représentation de longs textes est que leur concaténation est inefficace : concaténer deux chaîne commence par allouer un nouvel espace, puis copie la première dans ce nouvel espace, puis copie la deuxième. Quand on construit bout par bout un large document, il serait déraisonnable d'utiliser les `String` de Scala ; il faut changer de structure de donnée.

On se propose ici d'implémenter une structure de donnée arborescente immutable, représentée par la classe `Text`, pour manipuler du texte de façon efficace :

```
sealed abstract class Text
case class Atom(s: String) extends Text
case class Join(t1: Text, t2: Text) extends Text
```

Une valeur `Atom(s)` représente la chaîne de caractère `s` ; une valeur `Join(t1, t2)` représente la concaténation de la représentation de `t1` et de la représentation de `t2`.

Exemple 1. *La valeur :*

```
val t = Join(Join(Atom("hello"), Atom(" ")), Atom("world"))
```

représente la chaîne `"hello world"`.

L'intérêt de cette structure de donnée est que la concaténation de deux `Text` `t1` et `t2` est efficace (en $O(1)$) : il suffit de créer un nouveau noeud `Join(t1, t2)`. Notez cependant que cette représentation n'est pas *canonique* : deux valeurs différentes de type `Text` peuvent représenter la même chaîne de caractère.

Exemple 2. *La valeur :*

```
val u = Join(Atom("hell"), Join(Atom("o w"), Atom("orld")))
```

représente la même chaîne que `t`.

2.1 Représentation du texte pur

Nous allons étendre cette classe avec des méthodes similaires à celles de la class `String`, de façon à pouvoir la substituer de façon transparente à `String` dans un futur développement. Dans la suite, toutes les fonctions doivent être *immutables*, c'est-à-dire ne jamais modifier en place une structure de donnée.

1. Implémentez les méthodes (triviales donc) de concaténation :

   ```
   def +(t: Text): Text
   def +(s: String): Text
   ```

2. Implémentez la méthode `length(): Int` qui renvoit la longueur du `Text`
3. Implémentez la méthode `toString(): String` qui renvoit la chaîne de caractère représentée par le `Text`.
4. Réimplémentez la méthode `equals(t:Text): Boolean` qui teste l'égalité de deux `Text`. [1]
5. Implémentez la méthode `map(f: String=>String): Text` qui applique `f` à toutes les chaînes d'un `Text`. Grâce à cette dernière, implémentez la méthode `capitalize(): Text` qui met le text en majuscule. [2]
6. Implémentez la méthode `charAt(i:Int): Option[Char]` qui retourne le `i`-ème caractère du `Text` sur lequel elle est appliquée, ou `None` s'il n'y en a pas. Vous pourrez vous servir pour cela de la méthode de même nom et type sur les `String`. [3]

2.2 Texte formatté

On décide maintenant d'augmenter notre représentation de texte avec des indicateurs de mise en forme : **gras**, *italique*, ou ***les deux combinés***. Pour cela, on déclare deux nouveaux types de noeuds possibles dans notre arborescence ; les lignes suivantes sont ajoutées après les déclarations de `Atom` et `Join` :

```
case class Bold(t: Text) extends Text
case class Ital(t: Text) extends Text
```

Si `t` représente un `Text` quelconque, alors `Bold(t)` (respectivement `Ital(t)`) représente ce texte mis en gras (respectivement italique). Un noeud `Bold` imbriqué sous un autre noeud `Bold` n'a aucun effet, et de même pour `Ital`.

1. On ajoute les lignes ci-dessus, et on recompile le fichier tel quel ; quelle erreur afficherait alors le compilateur Scala ? Expliquez brièvement comment modifier le code précédent pour le rendre correct.
2. En particulier, modifier la méthode `toString` de façon à convertir les noeuds de mise en forme en les balises HTML `<b></b>` et `<i></i>` respectivement.
3. Implémenter les méthodes `unbold(): Text` et `unitalics(): Text` qui retirent du texte toute annotation de mise en forme.
4. Implémentez la méthode `normalize(): Text` qui retire les noeuds imbriqués qui n'ont aucun effet. Attention, cette méthode ne devra avoir aucun effet sur la représentation (le texte mis en forme) !

1. On parle évidemment de l'égalité de leur représentation. Vous avez plusieurs solutions, notamment : une simple, utilisant les méthodes déjà définies, et une, beaucoup plus complexe, travaillant directement sur la structure du `Text`. On ne vous demande que la simple bien sûr ; toute tentative complémentaire sera compté comme un bonus.

2. Vous utiliserez la méthode éponyme sur les `String`.

3. En réalité cette méthode lève une exception au lieu de renvoyer un `Option` ; pour les besoins de l'examen, simplifions un peu la réalité.