Programmation avancée

Examen intermédiaire

jeudi 19 novembre 2009

Nom : _____

Prénom : _____

Vos points sont *précieux*, ne les gaspillez pas !

Votre nom Le travail qui ne peut pas vous être attribué est perdu: écrivez votre nom sur chaque feuille que vous rendez.

Votre temps Tous les points ne sont pas égaux. En effet, nous ne pensons pas que tous les exercices ont la même difficulté, même s'ils ont le même nombre de points.

Votre attention La donnée de chaque exercice est précisément formulée, et parfois subtile. Si vous ne la comprenez pas, vous ne pourrez pas en tirer tous les points.

Exercice	**Points**	**Points obtenus**
1	10	
2	10	
3	10	
Total	30	

Exercice 1 : Preuve par induction (10 points)

Les méthodes `take(n: Int)` et `drop(n: Int)` de la classe `List` retournent les n premiers, respectivement derniers, élément de la liste. De l'implantation de la méthode `take`, on a pu déduire les axiomes suivants.

$$\begin{array}{rcll} \texttt{Nil take n} & = & \texttt{Nil} & (t_0) \\ \texttt{xs take 0} & = & \texttt{Nil} & (t_1) \\ \texttt{(x :: xs) take (n + 1)} & = & \texttt{x :: (xs take n)} & (t_2) \end{array}$$

L'implantation de `drop` est la suivante.

```
class List[A] { ...
  def drop(n: Int): List[A] = this match {
    case Nil => Nil
    case x :: xs => if (n > 0) xs drop (n - 1) else this
  }
}
```

Pour mémoire, les axiomes de `:::` sont les suivants.

$$\begin{array}{rcll} \texttt{Nil ::: xs} & = & \texttt{xs} & (c_0) \\ \texttt{(x :: xs) ::: ys} & = & \texttt{x :: (xs ::: ys)} & (c_1) \end{array}$$

Votre tâche pour cet exercice

1. Ecrivez les axiomes déduis de l'implantation de `drop`.
2. En vous servant des axiomes, démontrez la propriété suivante.

   ```
   (xs take n) ::: (xs drop n) = xs
   ```

   Votre preuve doit mentionner la variable d'induction, l'hypothèse d'induction et, pour chaque étape de transformation, l'axiome ou l'hypothèse utilisé.

Exercice 2 : Arrangements de table (10 points)

Des étudiants, ayant terminé un examen difficile, s'installent le long d'un bar linéaire. Ils souhaitent connaitre tous les arrangements possibles pour s'assoir de façon à ce que des étudiants qui se détestent ne soient pas assis l'un à côté de l'autre.

La fonction qui répond à cette question a la signature suivante.

```
type Students = List[String]
type Hate = List[(String, String)]
def bar(sits: Students, stands: Students, hate: Hate): List[Students]
```

Un étudiant est représenté par un nom unique. `sits` contient les étudiants qui sont déjà assis le long du bar (le dernier étudiant qui s'est assis est à `sits.head`) et qui ne changeront plus de place. `stands` contient les étudiants qui veulent encore s'assoir. Le paramètre `hate` contient une liste de pairs d'étudiants qui se détestent. Si Aristide déteste Brünhild et Billal, la liste `hate` pourrait contenir les entrées suivantes.

```
List(("Brunhild", "Aristide"), ("Aristide", "Billal"))
```

`bar` retourne la liste de tous les arrangements possibles après que tous les étudiants se sont assis. Un arrangement est une liste de noms d'étudiants dans laquelle deux noms d'étudiants qui se détestent ne se suivent pas.

Conseil Inspirez-vous de la solution au problème des n-reines, vu dans le cours n° 6 et se basant sur les compréhensions *for*.

Votre tâche pour cet exercice

1. Implantez une fonction annexe `canSit` qui retourne **`true`** si deux étudiants peuvent s'assoir l'un à côté de l'autre pour un `hate` donné (c'est à dire qui teste si les deux étudiants ne se détestent pas).
2. Implantez la fonction `bar` de façon récursive.

Exercice 3 : Palindromes (10 points)

Un palindrome est une chaine de caractères qui est identique lorsqu'elle est lue de gauche à droite ou de droite à gauche. Certains nombres sont des palindromes dans leur représentation décimale (base 10) ou dans leur représentation binaire (base 2). $585_{10} = 1001001001_2$ est un palindrome dans les deux représentations, contrairement à $121_{10} = 1111001_2$ qui ne l'est que dans sa représentation décimale.

Votre tâche pour cet exercice

1. Ecrivez une expression qui calcule la somme de tous les nombres compris entre 0 et 1000000 dont la représentation décimale et la représentation binaire sont des palindromes.

Vous pouvez écrire autant de fonctions annexes que vous le souhaitez. Il n'est pas nécessaire que votre algorithme soit efficace.