Université Paris Diderot
M1 Info

Programmation Fonctionnelle Avancée
Année 2014-2015

Examen

Jeudi, 15 janvier 2015

Tout document papier est autorisé. Les ordinateurs, les téléphones portables, comme tout autre moyen de communication vers l'extérieur, doivent être éteints. Le temps à disposition est de 3 heures.

Les exercices doivent être rédigés en fonctionnel pur : ni références, ni tableaux, ni boucles `for` ou `while`, pas d'enregistrements à champs mutables. Chaque fonction ci-dessous peut utiliser les fonctions prédéfinies (sauf indication contraire), et/ou les fonctions des questions précédentes.

Cet énoncé a 4 pages.

Exercice 1 Soit la signature suivante :

```
module type STRING =
sig
 type t (* type des chaines de caracteres *)
 val length : t -> int (* longueur d'une chaine de caracteres *)
 val get : t -> int -> char (* acces au n-eme caractere *)
 val sub : t -> int -> int -> t (* sous-chaine (position du debut, longueur) *)
 val (^) : t -> t -> t (* concatenation de chaines *)
 val print : t -> unit (* affichage d'une chaine *)
end
```

Le module `String` de la bibliothèque standard d'OCaml fournit une implémentation de cette signature `STRING`, pour laquelle le type `String.t` est le type de base `string`. Cette implémentation présente des limitations :

- La taille maximale d'une chaîne de type `string` est bornée[1].
- La concaténation de grandes chaînes peut devenir coûteuse en temps et entraîner des recopies de mémoire parfois indésirables.

Dans cet exercice, on se propose de coder une structure de données appelée *ropes* (ou *cordes*), qui permet de représenter efficacement de très grandes chaînes de caractères.

On définit le type `rope` ainsi :

```
type rope =
  | Sub of string * int * int
  | Cat of rope * rope * int
```

Dans cette représentation, une corde est :

- soit une portion d'une chaîne habituelle, délimitée par une position de départ et une longueur ;
- soit une concaténation de deux autres cordes, annotée par la longueur totale de ces deux cordes.

1. En Caml sur une machine 32-bits, on a `Sys.max_string_length = 16777211`.

Par exemple, la valeur

```
Cat (Cat (Sub ("abc",2,1), Sub ("def",1,2), 3), Sub ("ghi",0,2), 5)
```

est une représentation possible de la chaîne "cefgh" en corde.

1. Écrire en Caml une fonction `length : rope → int` donnant la longueur totale d'une corde (en caractères).
2. Écrire en Caml une fonction `concat : rope → rope → rope` permettant de concaténer deux cordes en temps constant. En particulier, votre fonction ne doit pas être récursive ni faire appel à des fonctions récursives.
3. Écrire en Caml des fonctions `of_string : string → rope` et `to_string : rope → string`. Pour `to_string`, on supposera que la corde à convertir est suffisemment courte pour pouvoir en faire une `string`.
4. Écrire en Caml une fonction `get : rope → int → char` permettant d'accéder au `n`-ème caractère d'une corde.
5. Écrire en Caml une fonction `sub : rope → int → int → rope` de telle sorte que l'appel `(sub r n len)` permette d'obtenir la sous-corde de r commençant au `n`-ème caractère et de longueur `len`. Cette fonction `sub` ne doit pas allouer de nouvelles `string`. Par exemple, si `x` est la corde représentant "cefgh" donnée au début de l'exercice, l'appel `sub x 2 3` doit donner le résultat `Cat (Sub("def",2,1), Sub("ghi",0,2), 3)`, ce qui est une représentation de la la chaîne "fgh".
6. En s'inspirant des questions précédentes, écrire un foncteur permettant de créer une structure de corde à partir de tout module de signature `STRING`. Ce foncteur devra cacher les détails de l'implémentation des cordes derrière une signature similaire à `STRING`, avec en plus des fonctions `of_string` et `to_string`.

Exercice 2 Dans une librairie, vous trouvez le fragment de code suivant.

```
module type ATOM =
sig
    type atom
    val mkInt : int -> atom
    val mkBool : bool -> atom
    val toString : atom -> string
    val double : atom -> atom
    val conjunction : atom -> atom -> atom
end

module Atom : ATOM = struct
type atom = I of int | B of bool

let mkInt (i: int) = I i
let mkBool (b: bool) = B b

let toString = function
 | I i -> string_of_int i
 | B b -> string_of_bool b

let double (v: atom): atom = match v with
 | I i -> I (2* i)
 | _ -> failwith "type mismatch"

let conjunction (v1: atom) (v2: atom): atom =
match (v1,v2) with
 | B b1, B b2 -> B (b1 && b2)
 | _ -> failwith "type mismatch"
end;;
```

Vous vous apercevez qu'on peut facilement se tromper, et tomber sur des exceptions à l'exécution, par exemple en écrivant

```
open Atom
let x = mkBool true;;
let y = double x;;
```

On vous demande d'utiliser le système de type pour éviter ces problèmes, et cela de deux façons :

1. modifiez le code, *en utilisant des types fantôme*, de telle sorte que les exceptions ne soient jamais levées à l'exécution.
2. modifiez le code, *en utilisant des GADT*, de telle sorte que l'on n'ait plus besoin de failwith dans le code.

Dans les deux cas, une utilisation erronée du code (comme sur l'exemple donné au dessus) déclenchera une erreur de typage. Les deux solutions demandées sont indépendantes.

Exercice 3 On s'intéresse à ce puzzle arithmétique celèbre :

$$\begin{array}{cccccc} & & s & e & n & d \\ + & & m & o & r & e \\ \hline = & m & o & n & e & y \end{array}$$

Une solution à ce puzzle est un triplet de tuples

$$((s,e,n,d),(m,o,r,e),(m,o,n,e,y))$$

où s,e,n,d,m,o,r,y sont des valeurs naturelles *différentes* entre 0 et 9, s et m différent de 0, telles que l'addition montrée ci-dessus est correcte.

1. En utilisant la compréhension de listes vue en cours, écrire une expression qui, quand évaluée, donne la liste de toutes les solutions de ce puzzle.
   Indication : Il convient de définir une variante de la fonction `range` vue en cours qui vous aide à assurer que toutes les variables ont des valeurs différentes.
2. Pour éviter la tache un peu fastidieuse d'assurer que les variables d'une compréhension de listes prennent des valeurs différentes, on veut maintenant définir une variante des compréhensions qui assure que toutes les variables prennent des valeurs différentes (pour simplifier nous supposons que toutes les variables ont le type `int`). Par exemple,

   ```
   [! (x,y) | x<-range 0 2; y<- range 0 2 !]
   ```

   doit donner le résultat

   $$[(0,1);(0,2);(1,0);(1,2);(2,0);(2,1)]$$

   On partant de la traduction de la compréhension de listes donnée en cours, proposez une traduction de cette variante de la compréhension.
3. Refaites la Question 1, mais en utilisant maintenant la variante des compréhensions définie à la Question 2.