Algorithmes combinatoires et monades

Objectifs

- Fonctions combinatoires sur les listes avec la monade ND.
- Différentes implantations de la monade ND : listes, itérateurs, etc.

Cette séance revient sur la séance du TP3 consacrée aux algorithmes liés aux problèmes combinatoires. Nous allons abstraire cette combinatoire à l'aide la monade "non-déterministe" ND et ses différentes implantations. Une première implantation classique à l'aide des listes (NDList) est définie dans le fichier fourni `tp8_etu.ml`. Elle est équivalente aux solutions proposées dans le TP3.

Pour tous les exercices combinatoires, les fonctions demandées seront définies à l'intérieur d'un module paramétré par une implantation de la monade additive ND : MONADE_PLUS, dont on rappelle ci-dessous la signature. Par la suite, un squelette des différentes implantations demandées de ND sera fourni, incluant une fonction print : t -> unit affichant les résultats calculés, à des fins de tests.

```
module type FONCTEUR =
  sig
    type 'a t
    val map : ('a -> 'b) -> ('a t -> 'b t)
  end
module type MONADE =
  sig
    include FONCTEUR
    val return : 'a -> 'a t
    val (>>=) : 'a t -> ('a -> 'b t) -> 'b t
  end
module type MONADE_PLUS =
  sig
    include MONADE
    val zero : 'a t
    val (++) : 'a t -> 'a t -> 'a t
  end
```

1 Fonctions combinatoires

▷ **Exercice 1 (Combinaisons)**

- *Reprendre la fonction* combinaisons : 'a list -> int -> 'a list list *du TP3 et introduire les fonctions de la monade* ND *dans le code, afin d'obtenir* combinaisons : 'a list -> int -> 'a list ND.t
- *Appliquer cette fonction à l'implantation fournie* NDList *et afficher le résultat.*

▷ **Exercice 2 (Permutations)**

- *Reprendre la fonction* insertions_partout : 'a list -> int -> 'a list list *du TP3, qui prend en argument un élément* e *et une liste* l*, et qui renvoie la liste des insertions à toutes les positions possibles de* e *dans* l*, afin d'obtenir* insertions_partout : 'a list -> int -> 'a list ND.t

— *Reprendre la fonction* permutations : 'a list -> 'a list list *du TP3, qui prend en argument une liste l, et qui renvoie les permutations de l, afin d'obtenir* permutations : 'a list -> 'a list ND.t

▷ **Exercice 3 (Partitions)** *Reprendre la fonction* partitions : int -> int list list *du TP3 qui prend en argument un entier strictement positif n et qui renvoie toutes les partitions possibles de n, afin d'obtenir* partitions : int -> int list ND.t

2 Itérateurs

On souhaite maintenant implanter la monade ND différemment de NDList. On cherche une solution itérative, i.e. où l'on peut récupérer les éléments un à un, à la demande. Ceci permet d'alléger l'empreinte mémoire des fonctions combinatoires proposées, en ne calculant pas à l'avance tous les éléments avant de les exploiter (on se contente de les afficher). On utilise le type 'a Flux.t vu précédemment pour représenter les ensembles.

▷ **Exercice 4** *Définir l'implantation* NDIter *se basant sur les flux. Appliquer les fonctions combinatoires précédentes à cette nouvelle implantation de la monade* ND*.*

3 Tirage aléatoire

Enfin, on veut introduire un tirage aléatoire des éléments au lieu de les engendrer tous. Ici, la notion d'égalité et les équations monadiques ont un sens moins fort. À cause du tirage aléatoire, dire qu'une expression est égale à une autre revient à dire que l'ensemble des exécutions possibles est le même pour les deux expressions. On utilise le type unit -> 'a option pour représenter les fonctions de tirage aléatoire, le type option permettant de modéliser un tirage effectué dans un ensemble vide de choix possibles. La fonction prédéfinie Random.bool: unit -> bool pourra être employée pour réaliser un tirage booléen. Un tel tirage a lieu dans l'implantation de l'opérateur a ++ b, qui représente un choix explicite entre a et b.

▷ **Exercice 5** *Définir l'implantation* NDRandom *se basant sur un tirage aléatoire. Appliquer les fonctions combinatoires précédentes à cette nouvelle implantation de la monade* ND*.*

La solution précédente, naïve, ne permet pas un tirage équiprobable. Ainsi, pour les permutations par exemple, moins une permutation mélange les données d'entrée, plus sa probabilité d'être tirée est grande. Pour rétablir un choix équitable dans les exercices combinatoires, on adjoindra à la fonction de tirage le nombre de choix possibles renvoyés par cette fonction. Le tirage induit par l'expression a ++ b devra être rendu équitable en comparant la taille de a et celle de b. La fonction prédéfinie Random.int: int -> int pourra être employée pour réaliser un tirage entier borné. On utilise le type int * (unit -> 'a) pour représenter le nombre de choix possibles et une fonction de tirage associée. Notez qu'en dehors des exercices combinatoires, la taille ne pourra plus être qu'une estimation, notamment si on applique des fonctions non injectives (à travers map par exemple).

▷ **Exercice 6** *Définir la nouvelle implantation* NDFairRandom*. Appliquer les fonctions combinatoires précédentes à cette nouvelle implantation.*