

Travaux Dirigés de Programmation Fonctionnelle

«Schémas de programmes»

I Schémas de réduction

1. (d'une liste)

Utiliser les schémas :

```
List.fold_left f a [b1; ...; bn] = f (... (f (f a b1) b2) ...) bn
List.fold_right f [a1; ...; an] b = f a1 (f a2 (... (f an b) ...))
```

pour définir les fonctions :

- (a) Concaténation des éléments d'une liste de chaînes de caractères.

Exemple : `concat_string ["a";"b";"c"] = "abc"`

- (b) Liste des nombres pairs d'une liste de nombres entiers (en préservant l'ordre des éléments dans la liste).

Exemple : `pairs [2;3;5;7;4] = [2;4]`

2. (d'un arbre)

On définit le type des arbres binaires à valeur aux noeuds comme suit :

```
type 'a arbre = Vide | Racine of 'a arbre * 'a * 'a arbre;;
```

Définir par une fonctionnelle le schéma `reduce` d'accumulation des éléments d'un arbre binaire. Le principe de ce schéma est de synthétiser un résultat à partir des résultats des 2 sous-arbres et de la valeur en racine.

Il prend en argument une fonction de type `'a * 'b * 'a -> 'a`, une valeur de type `'a` et un arbre de type `'b arbre`, où `'b` est le type des éléments de l'arbre et `'a` est le type du résultat.

Voici un exemple d'utilisation de ce schéma pour calculer la somme des éléments d'un arbre de nombres entiers :

```
# reduce (function (g,v,d) -> g+v+d) 0;;
- : int arbre -> int = <fun>
```

3. Utiliser le schéma `reduce` de la question précédente pour définir les fonctions :

- (a) Liste triée des éléments dans l'arbre.
(b) Parcours des éléments en profondeur d'abord.
(c) Parcours des éléments en largeur d'abord.

II Application d'une fonction à tous les éléments

1. (d'une liste)

Définir le schéma suivant :

$$\text{map } f \text{ [e1; ...; en]} = [f \text{ e1; ...; f en}]$$

à l'aide du schéma `List.fold_right`.

2. Utiliser le schéma `map` pour :

- (a) Définir une fonctionnelle `adjoint` qui ajoute un même élément en tête de toutes les listes dans une liste.

Exemple : `adjoint 1 [[]; [2]; [3;4]] = [[1]; [1;2]; [1;3;4]]`

- (b) En déduire une fonction `prefix` qui retourne tous les débuts de liste d'une liste.

Exemple : `prefix [1;2;3;4] = [[]; [1]; [1;2]; [1;2;3]; [1;2;3;4]]`

- (c) En déduire une fonction `scan` qui retourne la liste des sommes partielles d'une liste de nombres entiers.

Exemple : `scan [1;2;3;4] = [0;1;3;6;10]`

3. (d'un arbre)

Étendre le schéma `map` précédent aux arbres binaires à valeurs aux noeuds.

Définir une fonctionnelle à l'aide du schéma `reduce` de la question (I 2.).