

# Cours - Programmation Fonctionnelle

Louis Thevenet

## Table des matières

1. Types abstraits .....	2
2. Types fantômes .....	2
2.1. Définition .....	2
2.2. Utilité .....	2
3. Types non uniformes .....	2
3.1. Définition .....	2
3.2. Exemples .....	2
3.3. Usage .....	2
4. Types algébriques généralisés (GADT) .....	2

# 1. Types abstraits

1. Il se situe dans le monde mathématique
2. On a une fonction d'abstraction  $AF : C \rightarrow A$  avec  $C$  un type concret
  - $AF$  n'est pas implémentable mais à documenter
  - Surjective
  - Pas forcément injective
3. Une opération concrète est correcte si, quand elle respecte ses éventuelles préconditions, elle commute avec  $AF$  sur  $RI$  (domaine de définition de  $AF$ )

**RI** Invariant de représentation (souvent implémentable)

# 2. Types fantômes

## 2.1. Définition

Un type fantôme est un type paramétré :

1. dont au moins un des paramètres n'apparaît pas dans la définition des valeurs de ce type
2. dont la définition est abstraite par une signature

## 2.2. Utilité

- caractériser un état interne/caché (type-state)  $\leadsto$  plutôt pour du code impératif (ou monade d'état, cf. cours 6)
- ne pénalise pas l'exécution (zero cost abstraction)

# 3. Types non uniformes

## 3.1. Définition

Un type (récurif) non uniforme 'a t fait apparaître des instances différentes du paramètre dans sa définition, fonctions de 'a.

## 3.2. Exemples

- listes alternées:

```
type ('a, 'b) alt_list = | Nil | Cons of 'a * ('b, 'a) alt_list
```

- arbres binaires équilibrés:

```
type 'a perfect_tree = | Empty | Node of 'a * ('a * 'a) perfect_tree
```

## 3.3. Usage

- représenter des invariants de structure “descendants”
- meilleure spécification
- nécessite la “récursion polymorphe”

Il faut utiliser la **récursion polymorphe** par contre.

Par exemple :

```
let rec depth = function
| Empty -> 0
| Node (_, sub) -> 1 + depth sub;;
Error: This expression has type ('a * 'a) perfect_tree
but an expression was expected of type 'a perfect_tree
The type variable 'a occurs inside 'a * 'a

(* devient *)
(* chaque application de depth doit avoir un type universellement quantifié*)
(* littéralement : pour tout alpha, ce type, i.e. 'a . ('a perfect_tree)*)
let rec depth : 'a . 'a perfect_tree -> int = function
| Empty -> 0
| Node (_, sub) -> 1 + depth sub;;
val depth : 'a perfect_tree -> int = <fun>
```

# 4. Types algébriques généralisés (GADT)