Programmation Fonctionnelle : des concepts aux applications web (NFP119)

Types récursifs (algébriques)

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Certains types prédéfinis revisités

Le type `bool`

```
type bool = true | false
```

Le type `'a list`

```
type 'a list = [] | (::) of 'a * 'a list
```

Le type `'a option`

```
type 'a option = None | Some of 'a
```

Définir d'autres types de données

Types énumérés

```
# type colour = Blue | Green | Red;;
type colour = Blue | Green | Red
# Red;;
- : colour = Red
# let to_int c =
  match c with
  | Red -> 1
  | Blue -> 2
  | Green -> 3;;
to_int : colour -> int = <fun>
```

Types de données récursifs (polymorphes)

Arbres binaires (valeurs associées aux nœuds)

```
# type 'a tree = Empty | Node of 'a * 'a tree * 'a tree;;
type 'a tree = Empty | Node of 'a * 'a tree * 'a tree
# Empty;
- : 'a tree = Empty
# let t = Node (3, Empty, Empty);;
val t : int tree = Node (3, Empty, Empty)
# let rec height t =
  match t with
    | Empty -> 0
    | Node (_, l, r) -> 1 + (max (height l) (height r));;
val height : 'a tree -> int = <fun>
# height t;
- : int = 1
```

1 Arbres binaires (valeurs associées aux feuilles)

```
# type 'a ltree = Leaf of 'a | LNode of 'a ltree * 'a ltree;;
type 'a ltree = Leaf of 'a | LNode of 'a ltree * 'a ltree
# Leaf 3;;
- : int ltree = Leaf 3
# let t = LNode (Leaf 3, Leaf 4);;
val t : int ltree = LNode (Leaf 3, Leaf 4)
# let rec height t =
  match t with
    | Leaf _ = 1
    | LNode (l, r) = 1 + (max (height l) (height r));;
val height : 'a ltree -> int = <fun>
# height t;;
- : int = 2
```

Arbres binaires en Python 3.12 (type Tree)

```
type Tree[A] = Empty | Node[A]

@dataclass
class Empty:
    pass

@dataclass
class Node[A]:
    value: A
    left: Tree[A]
    right: Tree[A]
```

Arbres binaires en Python 3.12 (fonction heigth)

```python
def height[A](t: Tree[A]) -> int:
    match t:
        case Empty():
            return 0
        case Node(_, l, r):
            return 1 + max(height(l), height(r))
```

Arbres binaires en Java 21 (type Tree)

```
sealed interface Tree<A> {}

record Empty<A>() implements Tree<A> {}

record Node<A>(A value, Tree<A> left, Tree<A> right) implements Tree<A> {}
```

Arbres binaires en Java 21 (fonction `height`)

```
static <A> int height(Tree<A> t) {
    switch (l) {
        case Empty() -> {
            return 0;
        }
        case Node(var __, var l, var r) -> {
            return 1 + max(height(l), height(r));
        }
    }
}
```