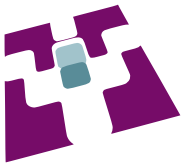
UNIVERSITÉ
CAEN
NORMANDIE

Autre Paradigme

Patrice Boizumault

Université de Caen - Normandie

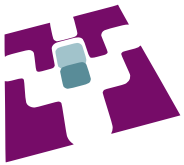
Lundi 03 février 2020



Arbres binaires

Terminologie

- issue de la botanique
  - racine, feuilles,
  - nœuds, branches
- issue des arbres généalogiques
  - fils, père, frères
  - ancêtres, ascendants, descendants

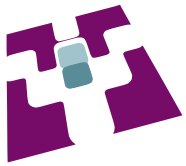


Arbres binaires

Plan :

1. **Arbres binaires non étiquetés**
2. Arbres binaires étiquetés
3. Typage : polymorphisme et pré-conditions
4. Arbres binaires de recherche (CM#6, TP#3)
5. Arbres de HUFFMAN[1] (CM#6, TP#4)

[1] https://fr.wikipedia.org/wiki/Codage_de_Huffman



Arbres binaires non étiquetés (Partie #1)

1er exemple

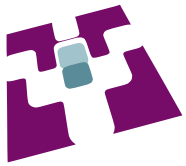
- définition par l'utilisateur à l'aide de la directive `data`[2]
- arbre binaire non étiqueté portant des entiers

```
data BinTree = Tip Integer | Bin BinTree BinTree
```

- 2 constructeurs pour le type utilisateur `BinTree`
  - `Tip` pour les feuilles (arité 1)
  - `Bin` pour les nœuds (arité 2)
- exemples

```
(Tip 5)
(Bin (Tip 1) (Tip 2))
(Bin (Bin (Tip 2) (Tip 4)) (Tip 6))
(Bin (Tip 5) (Bin (Bin (Tip 2) (Tip 4)) (Tip 6)))
```

[2] à ne pas confondre avec la directive `type`



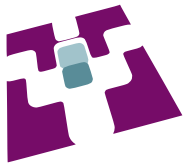
Arbres binaires non étiquetés

- Définition du type arbre binaire portant des entiers

```
data BinTree = Tip Integer
             | Bin BinTree BinTree
```

- Exemples

```
> see (Bin (Tip 1) (Tip 2))

    1
  o
    2

> see (Bin (Bin (Tip 2) (Tip 4)) (Tip 6))

        2
    o
        4
  o
    6
```



Arbres binaires non étiquetés

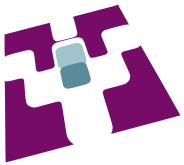
- Définition du type arbre binaire dont les feuilles sont des entiers

```
data BinTree = Tip Integer
             | Bin BinTree BinTree
```

- Exemples (suite)

```
> see (Bin (Tip 5) (Bin (Bin (Tip 2) (Tip 4)) (Tip 6)))
```

```
        5
    o
                2
            o
                4
        o
            6
```



Arbres binaires non étiquetés

Polymorphisme

- définition précédente

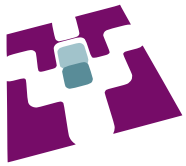
```
data BinTree = Tip Integer
             | Bin BinTree BinTree
```

- feuilles de type quelconque (représenté par la variable de type a)

```
data Tree a = Tip a
            | Bin (Tree a) (Tree a)
```

- exemples

```
(Bin (Tip 5) (Bin (Bin (Tip 2) (Tip 4)) (Tip 6)))
(Bin (Bin (Tip 2.7) (Tip 4.23)) (Tip 6.725))
(Bin (Bin (Tip 'a') (Tip 'b')) (Tip 'c'))
(Bin (Tip "as") (Bin (Tip "de") (Bin (Tip "zer") (Tip "p"))))
```

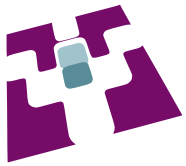


Arbres binaires non étiquetés

```
data Tree a = Tip a | Bin (Tree a) (Tree a)
```

```
> see (Bin (Bin (Tip 'a') (Tip 'b')) (Tip 'c'))

          'a'
     o
          'b'
o
     'c'
```

```
> see (Bin (Tip "as") (Bin (Tip "de") (Bin (Tip "zer") (Tip "p"))))

          "as"
o
          "de"
     o
               "zer"
          o
               "p"
```



Arbres binaires non étiquetés

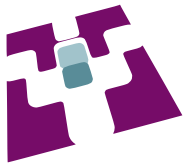
Traitement récursif

- Nombre de feuilles d'un arbre binaire

```
data Tree a = Tip a | Bin (Tree a) (Tree a)
```

- utilisation du *pattern-matching* : une équation pour les feuilles et une équation pour le traitement récursif des fils

```
nFeuilles :: Tree a -> Int

nFeuilles (Tip x)      = 1

nFeuilles (Bin t1 t2) = (nFeuilles t1) + (nFeuilles t2)


> nFeuilles (Tip "asde")  ==> 1

> nFeuilles (Bin (Bin (Tip 'a') (Tip 'b')) (Tip 'c'))  ==> 3
```



Arbres binaires non étiquetés

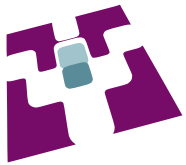
Traitement récursif

- nombre de noeuds d'un arbre binaire

```
data Tree a = Tip a | Bin (Tree a) (Tree a)
```

- utilisation du *pattern-matching*

```
nbNoeuds :: Tree a -> Int

nbNoeuds (Tip x)      = 0
nbNoeuds (Bin t1 t2) = 1 + (nbNoeuds t1) + (nbNoeuds t2)


> nbNoeuds (Tip "asde")   ==> 0

> nbNoeuds (Bin (Bin (Tip 'a') (Tip 'b')) (Tip 'c'))  ==> 2
```



Arbres binaires non étiquetés

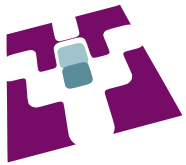
Traitement récursif

- profondeur d'un arbre binaire

```
data Tree a = Tip a | Bin (Tree a) (Tree a)

prof :: Tree a -> Int

prof (Tip x)      = 0
prof (Bin t1 t2) = 1 + max (prof t1) (prof t2)
```

- liste des feuilles d'un arbre binaire

```
lFeuilles :: Tree a -> [a]

lFeuilles (Tip x)      = [x]
lFeuilles (Bin t1 t2) = lFeuilles t1 ++ lFeuilles t2
```

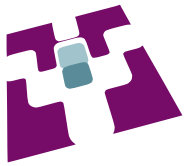


Arbres binaires non étiquetés

Traitement récursif

- liste des feuilles d'un arbre binaire

```
lFeuilles :: Tree a -> [a]

lFeuilles (Tip x)      = [x]
lFeuilles (Bin t1 t2) = lFeuilles t1 ++ lFeuilles t2
```

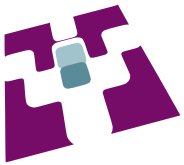


Arbres binaires non étiquetés

Traitement récursif

- visualiser un arbre en parenthèsant chaque niveau

```
data Tree a = Tip a | Bin (Tree a) (Tree a)

visu :: Tree a -> String

visu (Tip x) = show x

visu (Bin t1 t2) = "(" ++ visu t1 ++ " " ++ visu t2  ++ ")"


> visu (Bin (Bin (Tip 'a') (Tip 'b')) (Tip 'c'))
                                    ==> "(('a' 'b') 'c')"

> visu(Bin (Tip 5) (Bin (Bin (Tip 2) (Tip 4)) (Tip 6)))
                                    ==> "(5 ((2 4) 6))"
```

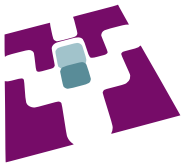


Arbres binaires non étiquetés

Fonctionnelles

- appliquer `f` à chaque feuille
- un arbre avec feuilles de type `a` devient un arbre avec feuilles de type `b`

```
data Tree a = Tip a
            | Bin (Tree a) (Tree a)

mapTree :: (a -> b) -> (Tree a) -> (Tree b)

mapTree f (Tip x)      = Tip (f x)
mapTree f (Bin t1 t2) = Bin (mapTree f t1) (mapTree f t2)

mapTree odd (Bin (Tip 15) (Bin (Bin (Tip 12) (Tip 14)) (Tip 16)))
     ==> (Bin (Tip True)
              (Bin (Bin (Tip False) (Tip False)) (Tip False)))
```

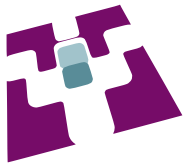


Arbres binaires

Plan :

1. *Arbres binaires non étiquetés*
2. **Arbres binaires étiquetés**
3. Typage : polymorphisme et pré-conditions
4. Arbres binaires de recherche (CM#6, TP#3)
5. Arbres de HUFFMAN[3] (CM#6, TP#4)

[3] https://fr.wikipedia.org/wiki/Codage_de_Huffman



Arbres binaires étiquetés (Partie #2)

Arbres binaires étiquetés

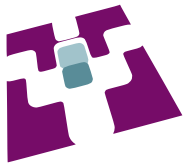
- jusque là, nos arbres ne portaient de l'information que sur les feuilles

```
data Tree a = Tip a
            | Bin (Tree a) (Tree a)
     deriving Show
```

- **Etiquetage**

```
data Etiq a b = Leaf a
              | Node b (Etiq a b) (Etiq a b)
     deriving Show
```

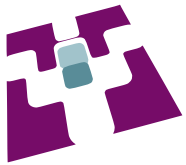
- exemples d'arbres étiquetés
  - les expressions arithmétiques
  - les formules propositionnelles



Arbres binaires étiquetés

Expressions arithmétiques

- ```
  data Etiq a b = Leaf a
                | Node b (Etiq a b) (Etiq a b)
       deriving Show
  ```
- ```
  > e1
  (Node "+" (Node "*" (Leaf 12) (Leaf 7))
            (Node "+" (Leaf 1) (Leaf 23)))
  > :t e1
  a :: Etiq Integer [Char]

  > e2
  (Node '+' (Node '*' (Leaf 12) (Leaf 7))
            (Node '+' (Leaf 1) (Leaf 23)))
  > :t e2
  b :: Etiq Integer Char
  ```

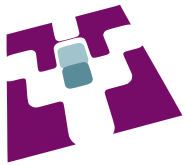


Arbres binaires étiquetés

```
data Etiq a b = Leaf a
              | Node b (Etiq a b) (Etiq a b)
```

- **visualiser en parenthèsant (parcours infixé)**

```
visuEtiq :: (Show a) => (Show b) => (Etiq a b) -> [Char]

visuEtiq (Leaf x) = show x
visuEtiq (Node n t1 t2) =
          "(" ++ visuEtiq t1
              ++ show n
              ++ visuEtiq t2 ++ ")"
> e2
(Node '+' (Node '*' (Leaf 12) (Leaf 7))
          (Node '+' (Leaf 1) (Leaf 23)))
> visuEtiq e2
"((12'*'7)'+'(1'+'23))"
```



Arbres binaires étiquetés

Fonctions dont le résultat est un arbre binaire

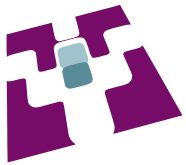
```
data Etiq a b = Leaf a  |  Node b (Etiq a b) (Etiq a b)
     deriving Show
```

- **transformer un** `(Etiq a b)` **en un** `(Tree a)`

```
e1 = Node 'o' (Leaf '_') (Node 'o' (Leaf '_') (Leaf '_'))
> unlabel e1
Bin (Tip '_') (Bin (Tip '_') (Tip '_'))
e2 = Node '+'(Node '*'(Leaf 2)(Leaf 7)) (Node '+'(Leaf 1)(Leaf
> unlabel e2
Bin (Bin (Tip 2) (Tip 7)) (Bin (Tip 1) (Tip 3))
```

En utilisant le *pattern-matching*

```
unlabel :: Etiq a b -> Tree a
unlabel (Leaf x) = Tip x
unlabel (Node _ t1 t2) =  Bin (unlabel t1) (unlabel t2)
```



Arbres binaires étiquetés

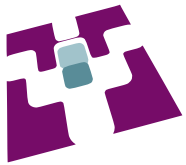
Fonctions dont le résultat est un arbre binaire

- transformer un `(Tree a)` en un `(Etiq a Int)` où chaque noeud indique le nombre de feuilles du sous-arbre dont il est la racine

  ```
  trans :: Tree a -> Etiq a Int
  ```

  en utilisant encore et toujours le *pattern-matching*

  ```
  trans (Tip x) = Leaf x

  trans (Bin t1 t2) = Node (nf t1' + nf t2') t1' t2'
                      where t1' = trans t1
                            t2' = trans t2
                            nf (Leaf _)     = 1
                            nf (Node n _ _) = n
  ```



Arbres binaires étiquetés

Fonctions dont le résultat est un arbre binaire

```
e4 = Node 'o' (Leaf '_') (Node 'o' (Leaf '_') (Leaf '_'))

> unlabel e4
Bin (Tip '_') (Bin (Tip '_') (Tip '_'))

> trans (unlabel e4)
Node 3 (Leaf '_') (Node 2 (Leaf '_') (Leaf '_'))


e2 = Node '+' (Node '*' (Leaf 12) (Leaf 7)) (Node '+' (Leaf 1) (Leaf 23))

> unlabel e2
Bin (Bin (Tip 12) (Tip 7)) (Bin (Tip 1) (Tip 23))

> (trans (unlabel e2))
Node 4 (Node 2 (Leaf 12) (Leaf 7)) (Node 2 (Leaf 1) (Leaf 23))


e3 =
Node '/' (Node '+' (Node '*' (Leaf 12) (Leaf 7)) (Node '+' (Leaf 1) (Leaf 23))) (Node '*' (Leaf 12) (Leaf 7))

> unlabel e3
Bin (Bin (Bin (Tip 12) (Tip 7)) (Bin (Tip 1) (Tip 23))) (Bin (Tip 12) (Tip 7))

> trans (unlabel e3)
Node 6 (Node 4 (Node 2 (Leaf 12) (Leaf 7)) (Node 2 (Leaf 1) (Leaf 23))) (Node 2 (Leaf 12) (Leaf 7))
```