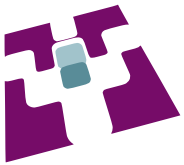
UNIVERSITÉ
CAEN
NORMANDIE

Autre Paradigme

Patrice Boizumault

Université de Caen - Normandie

Lundi 10 février 2020

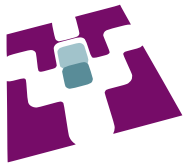


Arbres Binaires

Plan :

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

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



Typage et Polymorphisme (Partie #3)

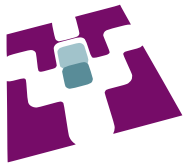
Exemple introductif

- La définition suivante n'est valide que s'il existe une relation d'égalité sur les éléments de type a.

```
myelem :: a -> [a] -> Bool
myelem x []       = False
myelem x (y:ys)   = (x==y) || (myelem x ys)

> No instance for (Eq a) arising from a use of ‘==’
    Possible fix:
      add (Eq a) to the context of
        the type signature for myelem :: a -> [a] -> Bool
```

- la signature (type) de `myelem` : `(Eq a) => a -> [a] -> Bool`
- préconditions
  - "le type a est une instance de la classe `Eq`"
  - "il existe une relation d'égalité pour le type a"



Typage et Polymorphisme

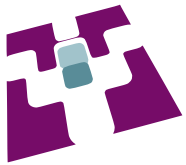
La classe `Eq`

- tout type a possédant une égalité ==a doit être déclaré comme une instance de la classe `Eq`,
- la fonction (méthode) ==a doit être définie,
- exemple : les entiers et les flottants où `integerEq` et `floatEq` sont des fonctions prédéfinies.

```
class Eq a where
        (==)    ::  a -> a -> Bool

instance Eq Integer where
        i1 == i2   = integerEq i1 i2

instance Eq Float where
        f1 == f2   = floatEq f1 f2
```



Typage et Polymorphisme

La classe `Ord`

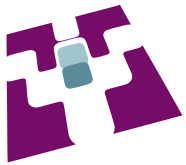
- la classe `Ord` est une sous classe de la classe `Eq`
- tout type `a` instance de `Ord` hérite des méthodes de la classe `Eq` (super classe)

```
class  (Eq a) => Ord a  where

     (<), (<=), (>), (>=)    : : a -> a -> Bool
     max, min                : : a -> a -> a

     ...
     max x y | x >= y    = x
             | otherwise = y

     min x y | x < y     = x
             | otherwise = y
    ...
```

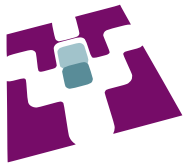


Typage et Polymorphisme

`deriving (Eq, Ord, Show, ...)`

- on définit un nouveau type `data t = ...`
- on ne veut pas tout redéfinir pour ce type
- on aimerait pouvoir hériter des classes *indispensables*
- déclarer que `t` est une instance dérivée de chacune de ces classes (`deriving`)

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



Typage et Polymorphisme

`deriving (Eq, Ord, Show, ...)`

- on hérite de l'égalité syntaxique

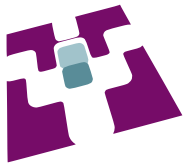
```
Tip 2 == Tip 2 => True
Tip 2 == Bin (Tip 2) (Tip 3) => False
```

- ordre sur les constructeurs : par arités croissantes, à égalité ordre lexicographique

```
Tip 2 < Tip 8 => True
```

```
Tip 2 < Bin (Tip 0) (Tip 3) => True
```

```
Bin (Tip 0) (Tip 3) < Bin (Tip 1) (Tip 1) => False
```



Typage et Polymorphisme

Pré-conditions d'utilisation

- (retour sur) visualiser un arbre en parenthèsant

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

- le type a doit posséder une méthode `show` (i.e. appartenir à la classe `Show`)

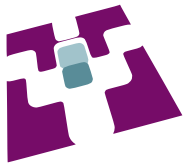
```
visu :: (Show a) => Tree a -> String

visu (Tip x) = show x
visu (Bin t1 t2) = "(" ++ visu t1 ++ " " ++ visu t2  ++ ")"
```

- Exemples

```
> 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))"
```



Typage et Polymorphisme

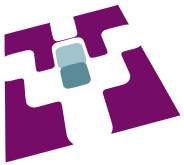
Conclusion

- un système de types polymorphes à la Hindley-Milner qui procure une sémantique pour le typage statique.
- la notion de classes de types permet d'y ajouter (proprement) la surcharge des opérateurs (overloading).

```
class Num a where
     (+)                 : : a -> a -> a
     negate              : : a -> a

instance Num Int where
     x+y                 = addInt x y
     negate x            = negateInt x

instance Num Float where
     x+y                 = addFloat x y
     negate x            = negateFloat x
```

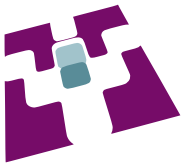


Arbres binaires

Plan :

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

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



Arbres binaires

Arbres binaires de recherche

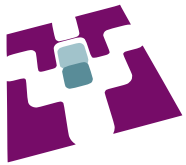
- définition :

```
data Btree a = Nil
             | Bin a (Btree a) (Btree a)
     deriving (Show, Ord, Eq)
```

- propriété : "les plus petits à gauche, les plus grands à droite"
- exemples

```
(Bin 4 (Bin 3 (Bin 2 Nil Nil) Nil)
    (Bin 7 (Bin 6 Nil Nil) (Bin 8 Nil Nil)))

(Bin 4 (Bin 3 (Bin 1 Nil Nil) Nil)
    (Bin 8 (Bin 7 Nil Nil) (Bin 11 (Bin 9 Nil Nil) Nil)))
```



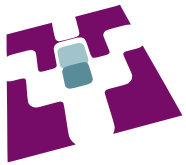
Arbres binaires

Arbres binaires de recherche

- visualiser un ABR sous forme indentée

```
a1 = (Bin 4 (Bin 3 (Bin 2 Nil Nil) Nil)
     (Bin 7 (Bin 6 Nil Nil) (Bin 8 Nil Nil)))

> voir a1
            2
      3
 4
            6
      7
            8
```



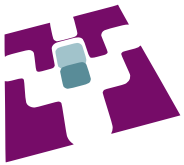
Arbres binaires

Arbres binaires de recherche

- visualiser un ABR sous forme indentée

```
a2 = (Bin 4 (Bin 3 (Bin 1 Nil Nil) Nil)
     (Bin 8 (Bin 7 Nil Nil) (Bin 11 (Bin 9 Nil Nil) Nil)))

> voir a2
            1
      3
 4
            7
      8
                  9
            11
```



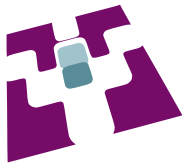
Arbres binaires

Arbres binaires de recherche

- visualiser un ABR sous forme indentée

```
data Btree a = Nil
             | Bin a (Btree a) (Btree a)
     deriving (Show, Ord, Eq)

visuTree :: Show a => Btree a -> String
visuTree t = f t 1
  where f Nil n = ""
        f (Bin y t1 t2) n = (f t1 (n+5)) ++
                            [' '|i <-[1..n]] ++ (show y) ++ ['\n']
                            ++ (f t2 (n+5))
```



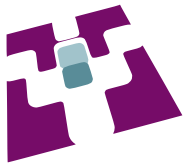
Arbres binaires de recherche

- **appartenance d'un élément à un ABR**

```
a1 = (Bin 4 (Bin 3 (Bin 2 Nil Nil) Nil)
     (Bin 7 (Bin 6 Nil Nil) (Bin 8 Nil Nil)))

> voir a1
            2
      3
 4
            6
      7
            8

> inBtree 2 a1    ==> True
> inBtree 7 a1    ==> True
> inBtree 9 a1    ==> False
```



Arbres binaires de recherche

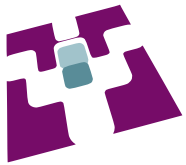
- **appartenance d'un élément à un ABR**

```
data Btree a = Nil
             | Bin a (Btree a) (Btree a)
     deriving (Show, Ord, Eq)


inBtree :: Ord a => a -> Btree a -> Bool

inBtree x Nil = False

inBtree x (Bin y t1 t2)
          | x < y     =
          | x > y     =
          | otherwise =
```



Arbres binaires de recherche

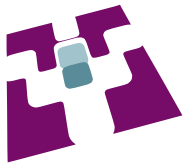
- **insertion d'un élément dans un ABR**

```
data Btree a = Nil
             | Bin a (Btree a) (Btree a)
     deriving (Show, Ord, Eq)


insere :: Ord a => a -> Btree a -> Btree a

insere x Nil = Bin x Nil Nil

insere x (Bin y t1 t2)
    | x < y     = Bin y ? ?
    | x > y     = Bin y ? ?
    | otherwise = ?
```

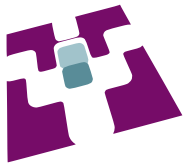


Arbres binaires de recherche

Exemple

```
                    > voir a2
          1
     3
4
          7
     8
               9
          11

                    > voir (insere 5 a2)
          1
     3
4
               5
          7
     8
               9
          11
```

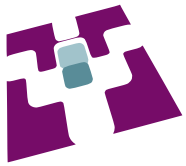


Arbres binaires de recherche

Exemple

```
                    > voir a2
          1
     3
4
          7
     8
               9
          11

                    > voir (insere 2 a2)
          1
               2
     3
4
          7
     8
               9
          11
```

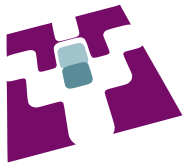


Arbres binaires de recherche

Exemple

```
                    > voir a2
          1
     3
4
          7
     8
               9
          11

                    > voir (insere 6 a2)
          1
     3
4
               6
          7
     8
               9
          11
```

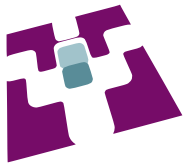


Arbres binaires de recherche

Exemple

```
                    > (list2abr [9, 5, 11, 7, 1, 8, 3, 4])
(Bin 4
    (Bin 3 (Bin 1 Nil Nil) Nil)
    (Bin 8
         (Bin 7 (Bin 5 Nil Nil) Nil)
         (Bin 11 (Bin 9 Nil Nil) Nil)))

                    > voir (list2abr [9, 5, 11, 7, 1, 8, 3, 4])
          1
     3
 4
               5
          7
     8
               9
          11
```



Arbres binaires de recherche

Exemple

```
                    > list2abr [1..8]
(Bin 8
     (Bin 7
          (Bin 6
               (Bin 5 (Bin 4 (Bin 3 (Bin 2 (Bin 1 Nil Nil) Nil) ...
               Nil)
          Nil)
     Nil)
                    > voir (list2abr [1..8])
                                   1
                              2
                         3
                     4
                5
           6
      7
 8
```