GRENOBLE 1 UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

INF121:
Algorithmique et Programmation Fonctionnelle
Cours 11 : Structures arborescentes

Année 2013 - 2014

Dans les précédents épisodes de INF121 . . .

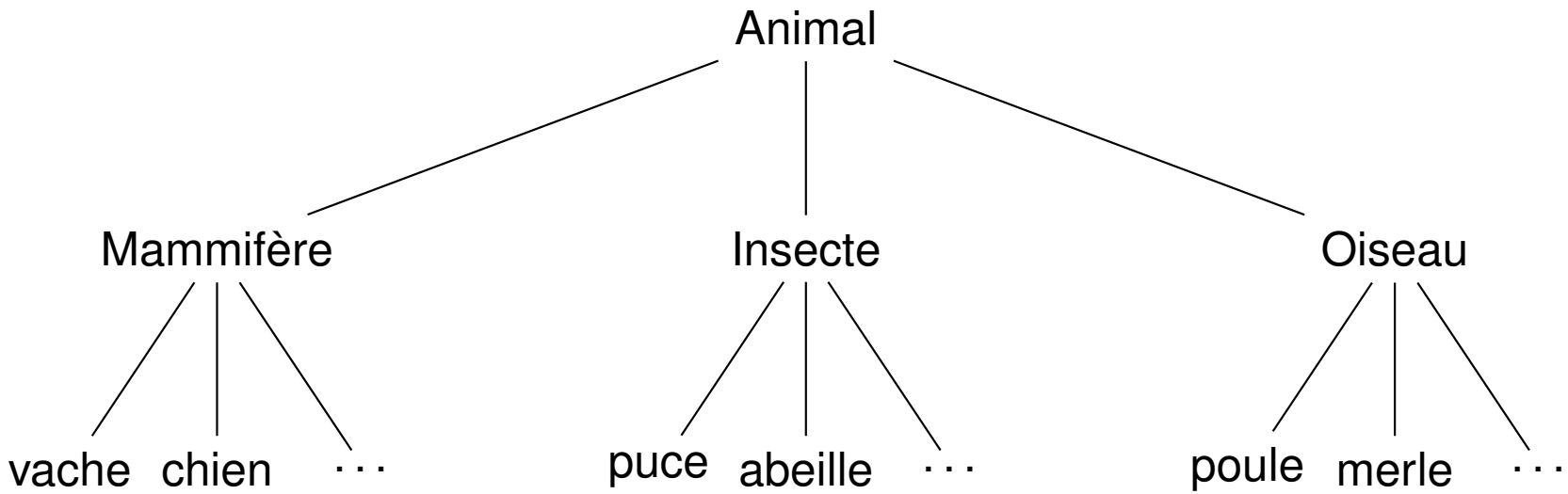
- Types de base : booléens, entiers, réels, etc.
- Identificateurs
- Fonctions
- Définitions de types : synonyme, énuméré, produit, somme
- Pattern-matching
- Récursivité
  - fonctions récursives
  - types récursifs
- polymorphisme
- ordre supérieur

Plan

A propos d'arbres (1)

intuition

Classification d'espèces :



Remarque

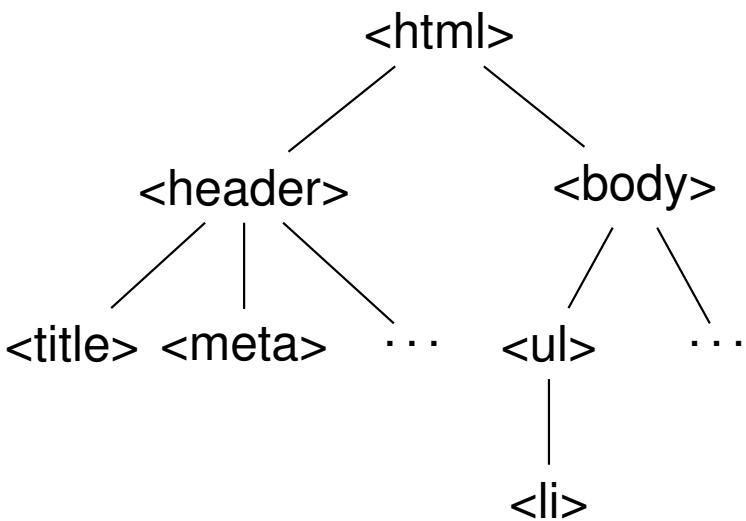
- noeuds avec étiquette, répétition possible d'étiquette
- noeud "racine", noeuds sans/avec "sous-arbres", noeud "père"
- structure hiérarchique
  - notion de niveau dans l'arbre
  - partition des noeuds en sous-arbres disjoints

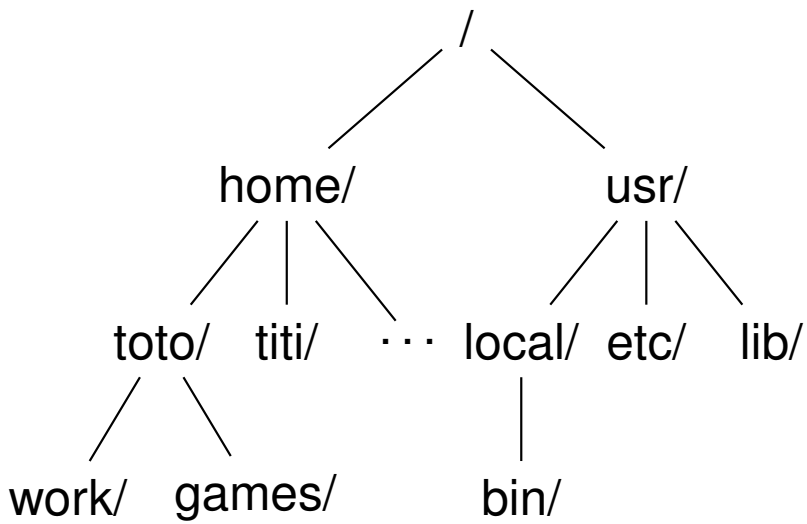
A propos d'arbres (2)

intuition

Intérêt : fournir une notion de hiérarchie (contrairement aux listes)

- facilite l'accès aux données
  (ex : système de fichiers, répertoires et sous-répertoires)
- permet de structurer l'information
  (ex : document HTML, organigramme, table des matières, etc.)
- permet de représenter des niveaux d'imbrications (parenthésage), ou des priorités (expressions arithmétiques)
- etc.



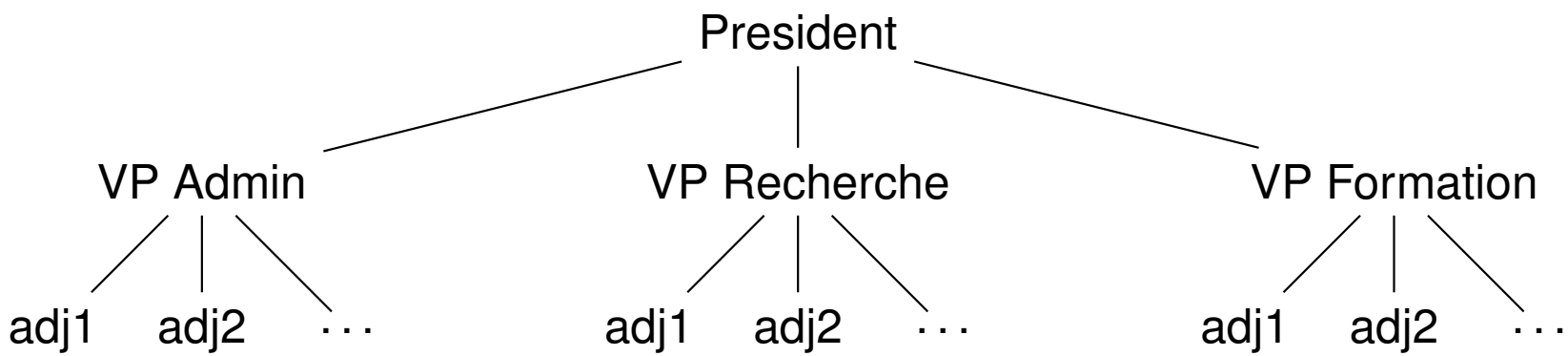


Arbre (étiqueté)

Un **arbre** est une structure récursive qui est :

- soit vide
- soit un noeud auquel est associé :
  - une étiquette
  - des fils : une séquence d'**arbres** (évent. vide)

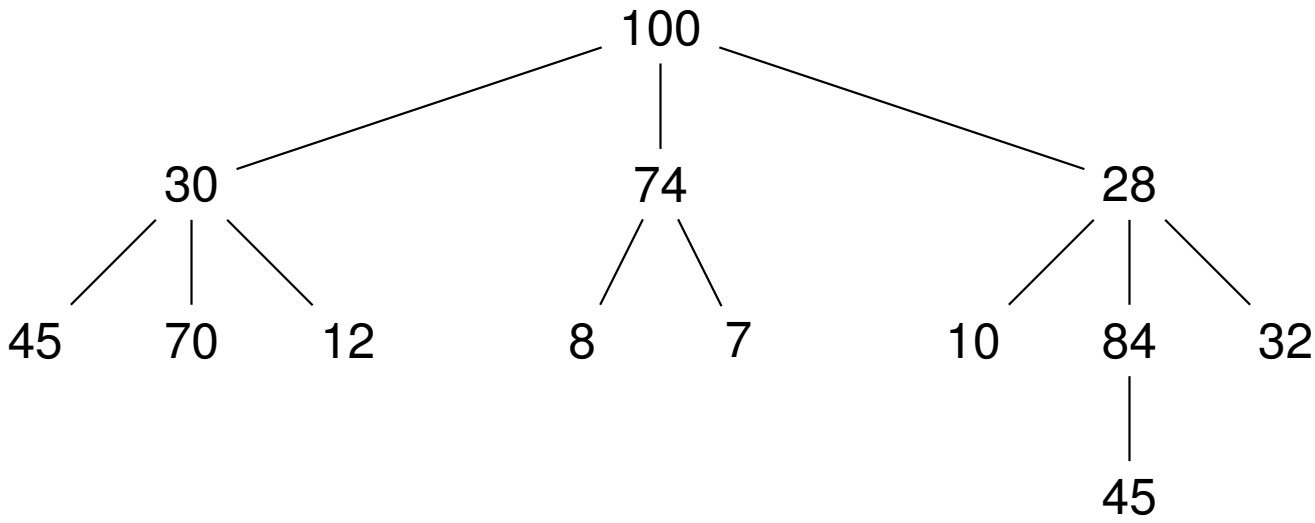
$\rightarrow$ permet de stocker des éléments (étiquettes) **de même type**



Vocabulaire

- Le noeud “le plus haut” est la racine
- La donnée associée ‘a un noeud est son étiquette / label / élément
- Les (sous-)arbres associés à un noeud sont ses fils, noeud père
- Un noeud sans fils est une feuille
- le chemin au noeud $n1$ est une séquence de noeuds père $\rightarrow$ fils allant de la racine à $n1$
- niveau d’un noeud : longueur (en nombre de noeud) du chemin à ce noeud
- hauteur (ou profondeur) d’un arbre : le niveau d’un noeud de niveau maximal
- taille d’un arbre : le nombre de noeuds qu’il contient

Exemple



- racine : 100
- étiquettes : 100, 30, 74, 28, 45, 70, 12, 8, 7, 10, 84, 32, 45
- feuilles : 45, 70, 12, 8, 7, 10, 45, 32
- fils du noeud 30 : 45, 70, 12
- 100 est le père de 30
- 100 est au niveau 1, 7 est au niveau 3
- la hauteur de l'arbre est 4
- [100;30;12] est le chemin au noeud d'étiquette 12

Plan

Arbres Binaires

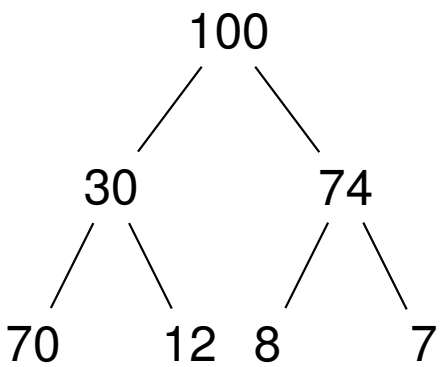
Définition et exemple

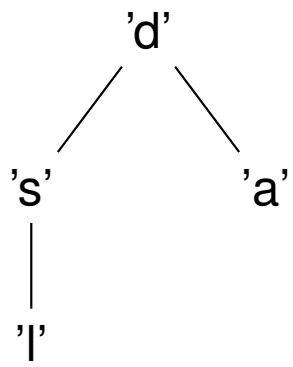
Un arbre est un arbre binaire si chaque noeud a *au plus* deux fils
Formellement :

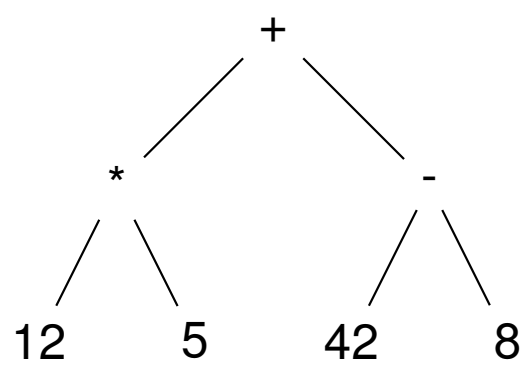
$$Abin(Elt) = \{Vide\} \cup \{Noeud(Ag, e, Ad) \mid e \in Elt \wedge Ag, Ad \in Abin(Elt)\}$$

Exemple : Arbre binaire sur des entiers

$$Abin(\mathbb{N}) = \{Vide\} \cup \{Noeud(Ag, e, Ar) \mid e \in \mathbb{N} \wedge Ag, Ad \in Abin(\mathbb{N})\}$$

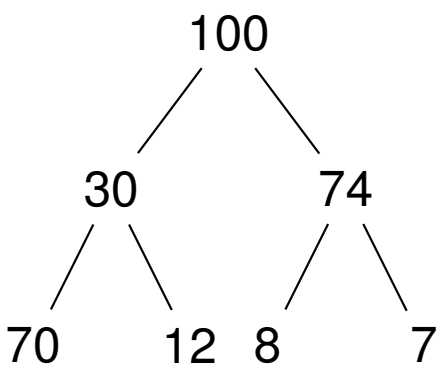


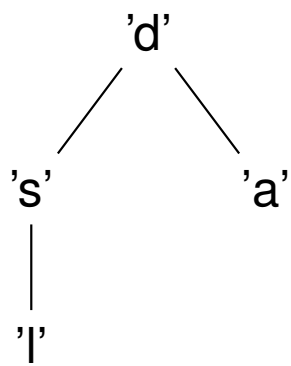


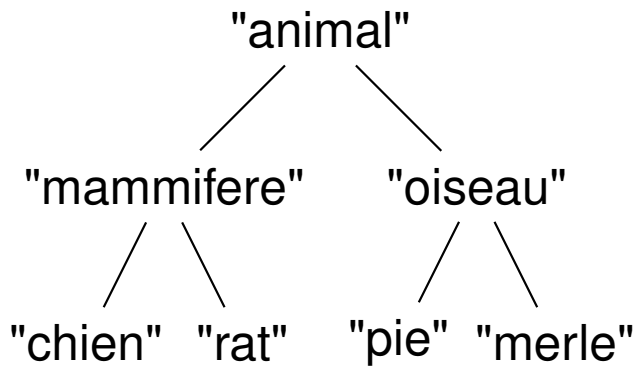


Vocabulaire

- Le premier (resp. second) fils est appelé fils gauche (resp. fils droit)
- Un arbre binaire a est complet ssi $\text{taille}(a) = 2^{\text{hauteur}(a)} - 1$







Arbres binaires d'entiers

En OCaml

Définir le type `arbre_binaire` ?
c'est un type somme, récursif, avec deux constructeurs :

- le constructeur `Vide` : l'arbre vide
  `Vide` $\in$ `arbre_binaire`
- le constructeur `Noeud` :
  ajout d'un noeud racine à partir d'une étiquette, d'un fils gauche et d'un fils droit
  `Noeud` $\in$ `etiq` $\times$ `arbre_binaire` $\times$ `arbre_binaire`

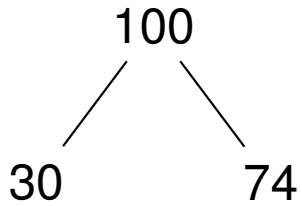
En OCaml :

```
type etiq = ... (* un type quelconque *)
type arbre_binaire =
  | Vide
  | Noeud of etiq * arbre_binaire * arbre_binaire
```

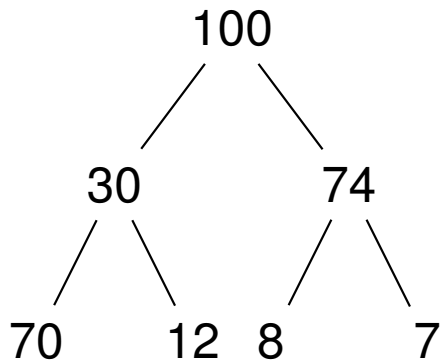
ou

```
type arbre_binaire
  | Vide
  | Node of arbre_binaire * etiq * arbre_binaire
```

Exemple d'éléments du type `arbre_binaire`




```
let ab1 =
   Noeud (100,
        Noeud (30,Vide,Vide),
        Noeud (74,Vide,Vide)
    )
```




```
let ab2 =
   Noeud(
    100,
    Noeud(30,
        Noeud(70,Vide,Vide),
        Noeud(12,Vide,Vide)
       ),
    Noeud(74,
        Noeud(8,Vide,Vide),
        Noeud(7,Vide,Vide)
       )
   )
```

Quelques fonctions (classiques) sur les arbres

Hauteur Le niveau maximal d'un noeud

```
let rec hauteur (a:arbre_binaire):int=
 match a with
  | Vide → 0
  | Noeud (_, a1, a2) → 1+ max (hauteur a1) (hauteur a2)
```

Exercices

Définir les fonctions suivantes :

- `somme` : renvoie la somme des éléments d'un arbres (d'entiers)
- `maximum` : renvoie l'élément maximal d'un arbre (d'entiers)

Arbres Binaires

... et polymorphisme

→ On peut paramétrer un arbre binaire par le type de ses éléments

```
type α arbre_binaire =
   | Vide
   | Noeud of α * α arbre_binaire * α arbre_binaire
```

Permet de définir plusieurs types “arbres binaires” :
`int arbre_binaire`, `char arbre_binaire`,
`string arbre_binaire`,...

DEMO: Définition d’arbres binaires

Appartient :
existence d'un élément de type α dans un α `arbre_binaire` ?

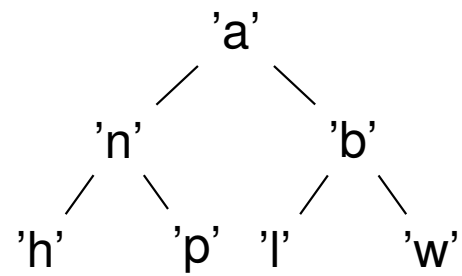
```
let rec appartient (elt:α) (a:α arbre_binaire):bool =
 match a with
  | Vide → false
  | Noeud (e,ag,ad) →
     (e=elt) || appartient elt ag || appartient elt ad
```

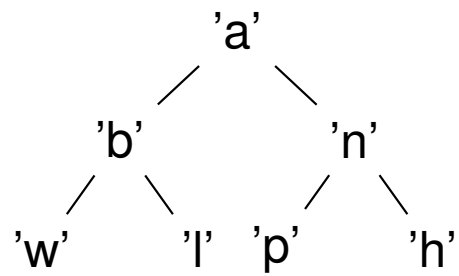
Liste des éléments d'un arbre :
Etant donné un α `arbre_binaire`, renvoie la α liste de ses éléments

```
let rec liste_elem (a:α arbre_binaire):α list=
 match a with
  | Vide → []
  | Noeud (elt,ag,ad) → (liste_elem ag)@(elt::(liste_elem ad))
```

Exercices : Définir les fonctions suivantes

- `taille`: nombre de noeuds d'un arbre binaire
- `feuilles`: liste des feuilles d'un arbre binaire
- `miroir`: image miroir d'un arbre binaire





Arbres binaires et ordre supérieur

On peut identifier plusieurs “schémas de fonction” sur les arbres :

- produire un nouvel arbre en appliquant une fonction à chaque noeud ($\sim$ opérateur **map**)
  - incrémenter toutes les étiquettes
  - remplacer chaque étiquette par la somme cumulée des étiquettes de ses fils

  `map` (`f` : $\alpha \to \beta$) (`a` : α `arbre_binaire`) : β `arbre_binaire` = ...

- produire un résultat en “accumulant” une valeur lors d’un parcours complet de tous les noeuds d’un arbre ($\sim$ opérateur **fold**)
  - nombre de noeuds, nombre de feuilles
  - liste des étiquettes

  `fold` (`f`:$\alpha \to \beta \to \beta \to \beta$) (`acc`:$\beta$) (`a`:$\alpha$ `arbre_binaire`) : β =

Différents ordres de parcours possibles d’un noeud `Noeud` (`elt`, `ag`, `ad`)

- traiter `elt`, puis parcourir `ag`, puis parcourir `ad` $\mapsto$ parcours prefixé
- parcourir `ag`, puis traiter `elt`, puis parcourir `ad` $\mapsto$ parcours infixé
- parcourir `ag`, puis parcourir `ad`, puis traiter `elt` $\mapsto$ parcours postfixé

Exemple d'opérateur "fold"

`fold_gauche_droite_racine`:
applique une fonction `f`

- à la racine
- et aux résultats obtenus (récursivement) sur les fils droit et gauche

```
let rec fold_gdr (f:α → β → β → β) (acc:β) (a:α arbre_binaire):β=
 match a with
  | Vide → acc
  | Noeud (elt, ag, ad) →
    let rg = fold_gdr f acc ag
      and rd = fold_gdr f acc ad
        in f elt rg rd
```

En utilisant la fonction `fold_gdr`, redéfinir les fonctions suivantes :

- `taille`
- `hauteur`
- `miroir`