

TD de *Logique et Circuits* n° 6
(Correction)

Définitions inductives: les arbres

Exercice 1 L'ensemble $\mathcal{A}(\mathbb{N})$ des arbres binaires étiquetées sur les entiers est la clôture inductive de l'ensemble des règles suivantes:

$$\frac{}{nil \in \mathcal{A}(\mathbb{N})} \qquad \frac{n \in \mathbb{N}, a_1 \in \mathcal{A}(\mathbb{N}) \text{ et } a_2 \in \mathcal{A}(\mathbb{N})}{cons(n, a, a') \in \mathcal{A}(\mathbb{N})}$$

Définir les fonctions *hauteur, taille* : $\mathcal{A}(\mathbb{N}) \rightarrow \mathbb{N}$.

Correction : ref: transparents de cours

Soit $\mathcal{R} \subseteq \mathcal{A}(\mathbb{N}) \times \mathcal{A}(\mathbb{N})$ le relation définie par $(a_1, a_2) \in \mathcal{R}$ si *hauteur*(a_1) < *hauteur*(a_2) ou *hauteur*(a_1) = *hauteur*(a_2) et *taille*(a_1) ≤ *taille*(a_2).

1. Montrer qu'il existe deux arbres a_1, a_2 tels que $(a_1, a_2) \in \mathcal{R}, (a_2, a_1) \in \mathcal{R}$, et $a_1 \neq a_2$.

Correction : deux arbres de même taille et même hauteur avec un étiquetage différent

2. Prouver que \mathcal{R} est un preordre.

3. Quel est l'ensemble ordonné canoniquement associé à \mathcal{R} ?

Correction : l'ensemble $S = \{(n, m) \in \mathbb{N} \times \mathbb{N} | n \leq m \leq 2^n - 1\}$

4. L'ordre de cet ensemble ordonné est-il total, bien fondé?

Correction :

- Oui, c'est un sous-ensemble d'un ensemble totalement ordonné
- c'est un ordre bien fondé, en effet:

Soit $((a_n, b_n))_{n \geq 0}$ une suite de S strictement décroissante et infinie, donc la suite $(a_n)_{n \geq 0}$ est décroissante (pas strictement) dans \mathbb{N} , d'où la suite $(a_n)_{n \geq 0}$ est stationnaire: il existe $n_0 \in \mathbb{N} \mid \forall n \geq n_0, a_n = a_{n_0}$. Par conséquent, la suite $((a_n, b_n))_{n \geq n_0}$ est égale à la suite $((a_{n_0}, b_n))_{n \geq n_0}$ et en plus strictement décroissante et infinie dans S , donc la suite $(b_n)_{n \geq n_0}$ est strictement décroissante et infinie dans \mathbb{N} , absurde.

Exercice 2 Calculer la clôture inductive des ensembles de règles suivantes:

- 1.

$$\frac{}{nil \in \mathcal{A}(\mathbb{N})} \qquad \frac{n \in \mathbb{N} \text{ et } a \in \mathcal{A}(\mathbb{N})}{cons(n, nil, a) \in \mathcal{A}(\mathbb{N})}$$

- 2.

$$\frac{}{nil \in \mathcal{A}(\mathbb{N})} \qquad \frac{a \in \mathcal{A}(\mathbb{N})}{cons(taille(a), a, a) \in \mathcal{A}(\mathbb{N})}$$

Correction :

1. les arbres peigne.
2. les arbres complets a dont l'étiquette à la racine est $2^{(hauteur(a)-1)} - 1$

Exercice 3 On considère le type Ocaml `type arbre = Av | Arb of int * arbre * arbre;;`

Un *parcours* d'un arbre est une fonction de type `arbre -> int list` qui renvoi la liste des étiquettes de son argument.

On définit trois parcours, en fonction de l'ordre dans lequel les actions suivantes sont exécutées:

- (a) Visiter la racine
- (b) Visiter le sous-arbre gauche
- (c) Visiter le sous-arbre droit

1. Parcours préfixé: a b c
2. Parcours infixé: b a c
3. Parcours postfixé: b c a

Définir une fonction pour chaque parcours.

Tester les trois fonctions sur `Arb(1, Arb (2, Av, Av), Arb (3, Av, Av))`.

Correction :

```
let rec prefixe a = match a with
Av -> []
| Arb(n,g,d)-> n::(List.append (prefixe g) (prefixe d));;

let rec infixe a = match a with
Av -> []
| Arb(n,g,d)-> let r= n::(infixe d) in List.append (infixe g) r;;

let rec postfixe a = match a with
Av -> []
| Arb(n,g,d)-> List.append (postfixe g)
(List.append (postfixe d) [n]);;
```

Exercice 4 Un arbre binaire de recherche (abr) est un arbre tel que:

pour chaque nœud $n(m,g,d)$, les étiquettes contenues dans le sous-arbre gauche g sont inférieures ou égales à m , et celles contenues dans le sous-arbre droit d sont supérieures à m .

Écrire une fonction `test : arbre -> bool` qui vérifie si un arbre est un abr.

Écrire une fonction `recherche : arbre -> int -> bool` testant la présence d'un élément dans un abr.

Correction :

```
let rec max a = match a with
Arb(n, _ , d) -> if d = Av then n else max d;;

let rec min a = match a with
Arb(n, g , _) -> if g = Av then n else min g;;

let rec test a = match a with
Av -> true
|Arb (n,Av,Av) -> true
|Arb (n,g,Av) -> (test g) && (max g) <= n
|Arb (n,Av,d) -> (test d) && (min d) > n
```

```

|Arb (n,g,d) -> (test d) && test(g) && (max g) <= n && (min d) >= n;;

(* solution geniale de Pado: *)

exception Error;;
let test a =
  let rec minmax a = match a with
    Arb(n,g,d) -> let min = if g=Av then n
                    else (let mmg = minmax g in
                          (if (snd mmg) > n then raise Error); fst mmg)
                    and max = if d=Av then n
                    else (let mmd = minmax g in
                          (if (fst mmd) <= n then raise Error); snd mmd)
                    in (min,max)
  in
  match a with
  Av -> true
  |_-> try let _= minmax a in true
        with Error -> false ;;

(* solution encore plus geniale de Francois*)

let test a =
let rec abr mi ma = function
Av -> true
| Arb(n,g,d)-> mi <=n && ma > n && abr mi n g && abr n ma d
in
abr minint maxint a

(*****)

let recherche a j =
  let rec cherche = function
    Arb (i, g, d) -> if i=j then
                        true
                      else
                        if j<i then cherche g else cherche d
    | Av -> false
  in cherche a

```