Démonstrations en informatique théorique

OPTION INFORMATIQUE - TP n° 2.6 - Olivier Reynet

A De l'intérêt des ensembles inductifs

A1. Écrire une fonction de signature `concat : 'a list -> 'a list -> 'a list` qui renvoie la concaténation de deux listes.

Solution :

```
let rec concat l1 l2 =
    match l1 with
    | [] -> l2
    | e::t -> e::(concat t l2);;
```

A2. Prouver la terminaison de la fonction `concat`.

Solution : Comme l'ordre sur les listes inductives est bien fondé (il n'existe par de suite strictement décroissante infinie), on peut procéder par récurrence. Lorsque la liste `l1` est vide (cas de base), l'algorithme s'achève immédiatement à la ligne `| [] -> l2` du filtrage de motif. Lorsque la liste n'est pas vide, on opère l'ajout en tête d'un élément à une liste (opération à coût constant) et un appel récursif sur une liste dont la taille est réduite de un élément. Ainsi, la suite des appels récursifs conduit nécessairement à la liste vide (cas de base) et s'achève.

A3. Prouver la correction de la fonction `concat`

Solution : On procède par induction structurelle sur les listes. Soit $\mathcal{P}$ la propriété : « La fonction `concat` est correcte».

(Cas de base) Dans le cas où la liste `l1` est vide, la concaténation d'une liste vide et d'une autre liste est égal à cette autre liste. La fonction renvoie donc bien le bon résultat.

(Règle de construction) Soit une liste `lst` pour laquelle la propriété est vérifiée, c'est-à-dire`concat lst l2` renvoie le bon résultat noté `l3`. Soit `e` un élément de liste et `lind` la liste construite en utilisant le constructeur ajouter en tête des listes de la manière suivante : `let lind = e :: lst`. Alors la fonction `concat lind l2` renvoie `e::(concat lst l2)`, c'est-à-dire, d'après notre hypothèse d'induction, `e::l3`, ce qui est le résultat correct.

(Conclusion) Comme $\mathcal{P}$ est vérifiée dans le cas de base et non modifiée par l'usage du constructeur ajouter en tête, $\mathcal{P}$ est donc vraie quelque soient les données d'entrée.

A4. Quelle est la complexité de la fonction `concat`?

Solution : La complexité $T(n)$ de la fonction `concat` vérifie la relation de récurrence suivante :

$$T(n) = 1 + T(n-1)$$

On reconnaît une suite artihmétique de raison 1 et de premier terme $T(0) = 1$. Donc, $T(n) = n+1 = O(n)$. Le complexité de la fonction `concat` est donc linéaire par rapport à la taille de la première opérande.

R D'une manière plus générale, si on dispose d'un ensemble inductif X défini de manière non ambigüe et d'une fonction f récursive sur X qui utilise la structure inductive de l'ensemble dans sa définition, alors :

- la terminaison de f est évidente car l'ordre induit sur X est bien fondé,
- la correction de f se montre par induction structurelle,
- la complexité s'analyse de la même manière qu'une fonction récursive quelconque.

B Arbres AVL

Afin de réduire au maximum les complexités des opérations, on cherche à minimiser la hauteur des arbres binaires de recherche utilisés pour les dictionnaires.

■ **Définition 1 — Arbre AVL.** Un arbre binaire de recherche est un arbre AVL [a] si, pour tout nœud, la différence entre la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit est $-1, 0$ ou 1 .

a. du nom de leur deux inventeurs : Adelson-Velsky et Landis

Un AVL garantit (cf. questions ci-dessous) que la hauteur de l'arbre est en $\Theta(\log n)$ où n est le nombre de nœuds.

■ **Définition 2 — Facteur d'équilibrage.** Le facteur d'équilibrage d'un nœud d'un arbre binaire de recherche est la différence entre la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit.

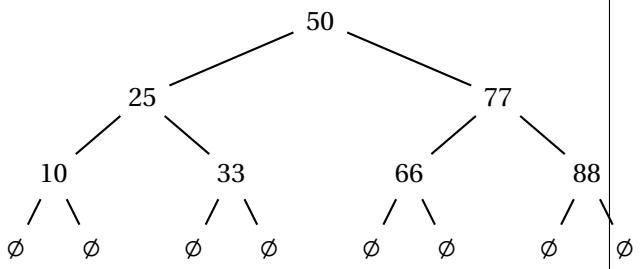
Ainsi, pour un arbre AVL, le facteur d'équilibrage de chaque nœud est -1 , 0 ou 1.

On souhaite montrer que la hauteur d'un arbre AVL est de l'ordre de $\log(n)$ où n est le nombre de nœuds. Pour cela, on note $(f_n)_{n\in\mathbb{N}}$ la suite de Fibonacci définie par :

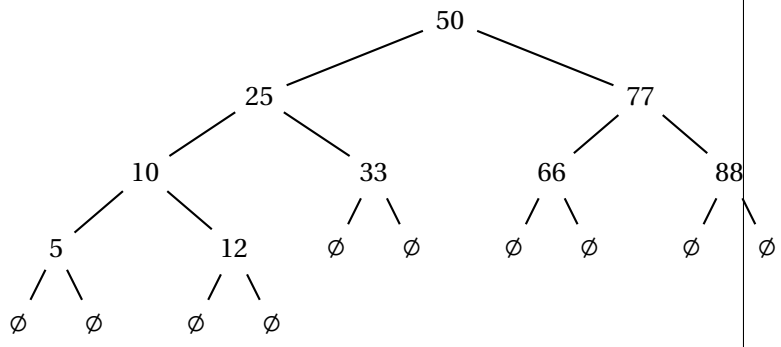
$$\begin{cases} f_0 = 1, f_1 = 1 \\ f_{n+2} = f_{n+1} + f_n \quad \text{pour tout } n \geqslant 0 \end{cases} \tag{1}$$

B1. Donner un exemple d'AVL pour les facteurs d'équilibrage 0, 1 et -1.

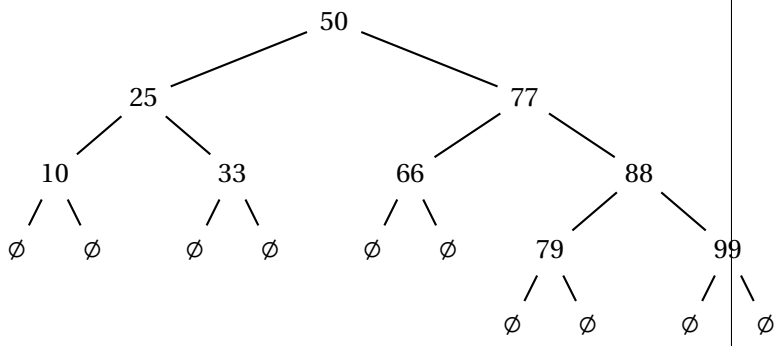
Solution : Pour un facteur d'équilibrage de 0 :



Pour un facteur d'équilibrage de 1 :



Pour un facteur d'équilibrage de -1 :



Les exemples ci-dessus sont également des arbres parfaits, ce qui n'est pas nécessaire pour la suite.

B2. Soit $h \in \{-1\} \cup \mathbb{N}$. Montrer qu'un arbre AVL de hauteur h possède au moins $f_{h+1} - 1$ nœuds.

Solution : On procède par induction structurelle. La propriété à démontrer est $\mathcal{P}$: un arbre AVL de hauteur h possède au moins $f_{h+1} - 1$ nœuds.

(Cas de base) Soit un arbre AVL vide. Sa hauteur vaut -1. Or, $f_{-1+1} - 1 = f_0 - 1 = 1 - 1 = 0$. Donc, $\mathcal{P}$ est vérifiée car l'arbre vide ne contient aucun nœud par définition.

(Pas d'induction) Soit deux arbres AVL g et d qui vérifient $\mathcal{P}$ et un arbre AVL a construit à partir de g et d. Le nombre de nœuds de a s'exprime

$$n_a = 1 + n_g + n_d \geqslant 1 + f_{h_g+1} - 1 + f_{h_d+1} - 1 = f_{h_g+1} + f_{h_d+1} - 1$$

. Trois cas sont alors possibles pour un AVL :

- $h_g = h_a - 1$ et $h_d = h_a - 2$: dans ce cas, $n_a \geqslant f_{h_a} + f_{h_a-1} - 1 = f_{h_a+1} - 1 \geqslant f_{h_a+1} - 1$,

- $h_g = h_a - 2$ et $h_d = h_a - 1$: dans ce cas, $n_a \geqslant f_{h_a-1} + f_{h_a} - 1 = f_{h_a+1} - 1 \geqslant f_{h_a+1} - 1$,
- $h_g = h_a - 1$ et $h_d = h_a - 1$: dans ce cas, $n_a \geqslant f_{h_a} + f_{h_a} - 1 \geqslant f_{h_a} + f_{h_a-1} - 1 = f_{h_a+1} - 1$,

Donc, l'arbre AVL a vérifie la propriété $\mathcal{P}$ dans tous les cas.

(Conclusion) Comme l'arbre vide vérifie $\mathcal{P}$ et que le constructeur de l'arbre binaire construit un arbre vérifiant $\mathcal{P}$ si les sous-arbres la vérifie, alors $\mathcal{P}$ est vraie pour tout arbre AVL.

Soit $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.62$ le nombre d'or. On rappelle que φ est racine du polynôme $X^2 - X - 1$.

B3. Montrer que $f_n \geqslant \varphi^{n-1}$ pour tout $n \in \mathbb{N}$.

Solution : On procède par récurrence double sur n :

- Pour $n = 0$, on a bien $\varphi^{n-1} = 1/\varphi \leqslant 1 = f_0$
- Pour $n = 1$, on a bien $\varphi^{n-1} = 1 \leqslant 1 = f_1$
- Soit $n \geqslant 2$. On suppose la propriété vraie aux rangs $(n-1)$ et $(n-2)$, et on la montre au rang n. En utilisant l'hypothèse de récurrence et le fait que φ est racine du polynôme $X^2 - X - 1$, on obtient :

$$f_n = f_{n-1} + f_{n-2} \geqslant \varphi^{n-2} + \varphi^{n-3} = \varphi^{n-3}(\varphi + 1) = \varphi^{n-3}\varphi^2 = \varphi^{n-1}.$$

- Conclusion : la propriété est vérifiée pour 0 et pour 1 et l'héridité est démontrée. Donc, par récurrence double, la propriété est vraie pour tout $n \in \mathbb{N}$.

B4. On considère un arbre binaire de recherche de hauteur h avec n nœuds. Montrer que si cet arbre est un AVL, alors $h \leqslant \log_\varphi(n+1)$.

Solution : D'après les questions précédentes :

$$n \geqslant f_{h+1} - 1 \geqslant \varphi^h - 1$$

D'où :

$$h \leqslant \log_\varphi(n+1)$$

B5. La réciproque de la proposition précédente est-elle vraie ?

Solution : La réciproque est fausse. Par exemple, l'arbre `Node(100, Node(50, Node(10, Empty, Empty), Empty), Empty)` n'est pas un AVL. Pourtant, $h = 2 = \log_2(4) \leqslant \log_\varphi(4) = \log_\varphi(n+1)$.

B6. Montrer que pour tout arbre binaire, $h \geqslant \log_2(n+1) - 1$.

Solution :

Le nombre maximal de nœud dans un arbre est atteint lorsque l'arbre est complet. Chaque niveau de l'arbre de hauteur k possède alors 2^k nœuds. On a donc, pour un arbre de hauteur h :

$$n_{max} = 1 + 2 + \ldots + 2^h = 2^{h+1} - 1 \tag{2}$$

On en déduit que $h \geqslant \log_2(n_{max}+1)-1$ et que donc $h \geqslant \log_2(n+1)-1$ pour tout arbre binaire, car qui peut le plus peut le moins.

B7. En conclure que la hauteur d'un arbre AVL est en $\Theta(\log n)$.

Solution : Pour un AVL on a donc $h = \mathcal{O}(\log n)$ et $h \geqslant \log_2(n+1)-1$, c'est-à-dire $h = \mathcal{O}(\log n)$ et $h = \Omega(\log n)$, ce qui signifie $h = \Theta(\log n)$.

C Arbres de Braun

Si t est un arbre, on note $|t|$ sa taille.

■ **Définition 3 — Arbre de Braun.** Un arbre de Braun est un arbre binaire qui est :

- soit l'arbre vide.
- soit de la forme $N(r,g,d)$ avec r une étiquette et g et d deux arbres de Braun vérifiant

$$|d| \leq |g| \leq |d|+1$$

On se limite dans ce sujet sujet au cas où les étiquettes des arbres de Braun sont des entiers et on représente de tels arbres à l'aide du type suivant en OCaml :

```
type braun = Vide | Noeud of int * braun * braun
```

C1. Pour tout $n \in [\![1,6]\!]$, dessiner les squelettes des arbres de Braun de taille n. Que remarque-t-on ?

Solution : On constate que pour tout $n \in [\![1,6]\!]$ il y a un unique arbre de Braun de taille n. C'est en fait vrai pour tout entier (démonstration par récurrence forte).

C2. Montrer que pour tout entier naturel n, il existe un arbre de Braun unique de taille n.

Solution : On procède par récurrence forte sur n ou induction structurelle.

Démonstration. Par induction structurelle.

(Cas de base) L'arbre vide est un arbre de Braun et est unique.

(Pas d'induction) Soient g et d deux arbres de Braun qui vérifient la propriété d'unicité. On construire a l'arbre de Braun `Noeud(e, g, d)` dont le nombre de nœuds est n. Nécessairement, $|g| = \lceil n/2 \rceil$ et $|d| = \lfloor n/2 \rfloor$. Or, d'après notre hypothèse d'induction, g et d sont uniques. Donc, a est unique au niveau structurel.

(Conclusion) Comme la propriété d'unicité est vérifiée pour le cas de base et n'est pas modifiée par la règle de construction, un arbre de Braun de taille n est unique.

■

Démonstration. par récurrence forte.

(Initialisation) Pour $n = 0$, il n'existe qu'un seul arbre de Braun, l'arbre `Vide`.

(Hérédité) Supposons l'unicité vraie pour tous les arbres de taille inférieure à n. Soit a un arbre de Braun de taille n. Il n'y a qu'une solution possible : son fils gauche est un abre de Braun de taille $|\, g \,| = \lceil n/2 \rceil$ et son fils droit un arbre de Braun de taille $|\, d \,| = \lfloor n/2 \rfloor$. Comme g et d sont uniques par hypothèse de récurrence, a est unique au niveau structurel.

(Conclusion) L'unicité est vérifiée pour $n = 0$ et on a démontré l'hérédité. Donc tout arbre de Braun est unique.

■

On dispose de la fonction suivante :

```
let rec add_braun x t =
    match t with
    |Vide -> Noeud(x, Vide, Vide)
    |Noeud(y,g,d) -> Noeud(y,add_braun x d,g);;
```

C3. Montrer que la fonction `add_braun` est correcte, c'est-à-dire qu'elle renvoie un arbre de Braun.

Solution : Procéder par induction structurelle.