Dans ce chapitre, on étudie trois structures inductives appelées "arbres", qui sont mathématiquement proches. On se concentre d'abord sur les *arbres binaires*, qui sont la structure la plus couramment utilisée et la plus simple à définir, puis on étudie deux variantes, les *arbres binaires stricts non vides* (ABSNV) et les *arbres d'arité quelconque*.

1 Arbres binaires

1.1 Définitions et propriétés

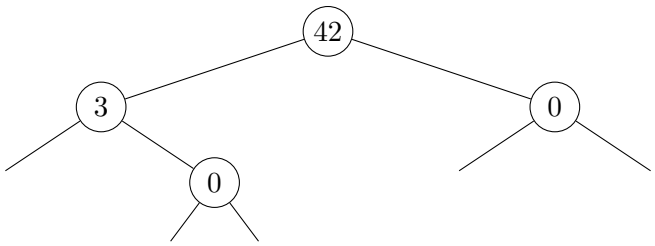
Définition 1. *Soit E un ensemble non vide. L'ensemble des arbres binaires d'étiquettes E est la structure inductive dont la signature est formée :*

- *d'un constructeur constant, noté* `Vide`*, et appelé arbre vide ;*
- *d'un constructeur d'arité 2, typé par E, noté* `Noeud`*.*

En OCaml, on aura donc :

```
type 'a arbre_bin = Vide | Noeud of 'a * 'a arbre_bin * 'a arbre_bin
```

Exemple 2. *$A =$* `Noeud (42, Noeud (3, Vide, Noeud (0, Vide, Vide)), Noeud (0, Vide, Vide))` *sera représenté graphiquement ainsi :*



Définition 3. *Soit A un arbre binaire étiquetté par E.*

Les sous-arbres *de A sont les arbres binaires étiquettés par E qui sont inférieurs ou égaux à A selon l'ordre structurel.*

Les nœuds *de A sont les arbres binaires non vides qui apparaissent dans l'écriture de A, avec leur position : deux nœuds différents de A peuvent donc correspondre à un même sous-arbre.*

Si A est non vide, le nœud de A qui correspond à A tout entier est appelé la racine *de A.*

*L'*étiquette *d'un nœud est l'élément de E correspondant au constructeur* `Noeud` *de ce nœud.*

Exemple 4. *Avec l'arbre A plus haut on a donc quatre sous-arbres distincts :*

- *A*
- `Noeud (3, Vide, Noeud (0, Vide, Vide))`
- `Noeud (0, Vide, Vide)`
- `Vide`

A a également quatre nœuds : sa racine (étiquettée par 42), un nœud étiquetté par 3, et deux nœuds étiquettés par 0.

Définition 5. *Soit A un arbre binaire non vide. Les sous-termes immédiats de A sont appelés son* sous-arbre gauche *(SAG) et* sous-arbre droit *(SAD).*

Définition 6. *Soit A un arbre binaire.*

La taille *de A est son nombre de nœuds.*

La hauteur *de A est la quantité définie récursivement par :*

$$\begin{cases} h(\texttt{Vide}) = -1 \\ \forall x \in E, A_g, A_d \text{ des arbres binaires}, h(\texttt{Noeud}(x, A_g, A_d)) = 1 + \max\Big(h(A_g); h(A_d)\Big) \end{cases}$$

Exemple 7. *Dans les exemples précédents, A est de hauteur 2 et de taille 4.*

Remarque 8. *La taille d'un arbre binaire non vide $A = \texttt{Noeud}(x, A_g, A_d)$ est $1 + \text{taille}(A_g) + \text{taille}(A_d)$.*

Proposition 9. *Soit A un arbre binaire, n sa taille et h sa hauteur.*

Alors $h + 1 \leqslant n \leqslant 2^{h+1} - 1$.

De plus, pour tout $h \in [\![-1; +\infty[\![$, il existe un arbre binaire de hauteur h et de taille $h + 1$, et un arbre binaire de hauteur h et de taille $2^{h+1} - 1$.

Preuve Montrons les inégalités par induction structurelle sur les arbres binaires.

- *Init :* L'arbre vide est de taille 0 et de hauteur -1.
  On a bien $-1 + 1 \leqslant 0 \leqslant 2^{-1+1} - 1$.
- *Hérédité :* Soit $A = \texttt{Noeud}(x, A_g, A_d)$ un arbre binaire non vide de hauteur $h \geqslant 0$ et de taille $n \geqslant 1$. Soient $h_g \geqslant -1$ et $h_d \geqslant -1$ les hauteurs de A_g et A_d, et n_g et n_d leurs tailles. On suppose que A_g et A_d vérifient la propriété.
  Par définition, $h = 1 + \max(h_g; h_d)$. On a également remarqué que $n = 1 + n_g + n_d$.
  Or par hypothèse d'induction, on a $h_g + 1 \leqslant n_g \leqslant 2^{h_g+1} - 1$ et $h_d + 1 \leqslant n_d \leqslant 2^{h_d+1} - 1$.
  De plus comme h_g et h_d sont minorés par -1, $\max(h_g; h_d) + 1 = \max(h_g + 1; h_d + 1) \leqslant h_g + h_d + 2$.
  Donc en particulier $h + 1 = \max(h_g; h_d) + 2 \leqslant h_g + h_d + 3 \leqslant n_g + n_d + 1 = n$.
  De même, $2^{h+1} = 2 \times 2^{\max(h_g+1; h_d+1)} = 2 \times \max\left(2^{h_g+1}; 2^{h_d+1}\right) \geqslant 2^{h_g+1} + 2^{h_d+1}$.
  Donc en particulier $2^{h+1} - 1 \geqslant \left(2^{h_g+1} - 1\right) + \left(2^{h_d+1} - 1\right) + 1 \geqslant n_g + n_d + 1 = n$.

On a donc montré l'inégalité.

On peut montrer qu'elle est atteinte pour tout h par une récurrence immédiate, avec des arbres comme sur la figure 1 d'une part avec un *peigne*, qui est définit pour $h \geqslant 0$ par un SAG vide et un SAD peigne de hauteur $h - 1$, ou réciproquement, d'autre part avec un *arbre parfait*, qui pour $h \geqslant 0$ est défini par un SAG et le SAD qui sont des arbres parfaits de hauteur $h - 1$.

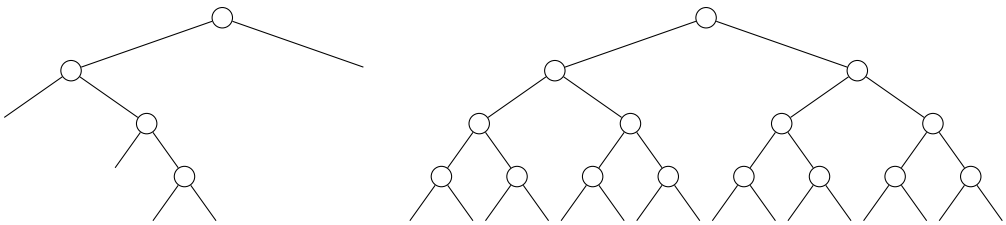


FIGURE 1 – Un peigne de hauteur 3 (il en existe 8, sans compter les étiquettes) et un arbre parfait de hauteur 3 (unique aux étiquettes près).

Définition 10. *Soit A un arbre binaire, N et M deux nœux de A.*

On dit que M est le parent [1] *de N, et que N est un* enfant [2] *de M, si N est le sous-arbre droit ou le sous-arbre gauche de M.*

Remarque 11. *Si A est un arbre binaire, tout nœud de A autre que la racine a un unique parent. La racine n'admet aucun nœud parent.*

Tout nœud de A a au plus deux enfants. On appelle parfois feuille *un nœud sans enfant (c'est à dire, dont le SAG et le SAD sont tous deux l'arbre vide), par analogie avec les arbres binaires stricts non vides (voir partie 2.*

1. Ou *père*, mais c'est un langage inutilement genré. En anglais, on utilise exclusivement *parent*.
2. De même, on rencontre aussi *fils* en français. L'anglais utilise exclusivement *child*.

Définition 12. *Soit A un arbre binaire, N un nœud de A.*

On appelle profondeur *de N le nombre de fois où on doit remonter au nœud parent en partant de N pour atteindre la racine.*

Proposition 13. *Soit A un arbre binaire, N un nœud de A.*

La profondeur de N est bien définie et est majorée par la hauteur de A.

Preuve Montrons le par induction structurelle sur les arbres binaires.

- *Init :* Si A est vide, il n'a aucun nœud, donc tous ses nœuds ont bien une profondeur bien définie et majorée par -1.
- *Hérédité :* Soit $A = \texttt{Noeud}(x, A_g, A_d)$ un arbre binaire non vide de hauteur $h \geqslant 0$. On suppose que pour tout N nœud de A_g, la profondeur de N dans A_g est bien définie et majorée par h_g la hauteur de A_g, et de même pour tout nœud de A_d.

  Soit N un nœud de A. Si N est la racine de A, sa prodondeur est $0 \leqslant h$.

  Sinon, N correspond soit à un nœud de A_g soit à un nœud de A_d. Par symétrie, on suppose SPDG qu'il existe N' nœud de A_g qui correspond à N.

  Si p est la profondeur de N dans A et p' celle de N' dans A_g, on a $p = p' + 1$ puisqu'il faut encore remonter exactement une fois depuis la racine de A_g.

  Or par hypothèse d'induction, $p' \leqslant h_g$. De plus, $h = 1 + \max(h_g; h_d)$. Donc $p = 1 + p' \leqslant h$.

Définition 14. *Soit $h \geqslant -1$ entier.*

On appelle parfait *un arbre binaire de hauteur h si tous ses nœuds de profondeur strictement inférieure à h ont exactement deux enfants*[1]

On appelle complet *un arbre binaire de hauteur h si tous ses nœuds de profondeur strictement inférieure à $h-1$ ont exactement deux enfants, et que le niveau de profondeur h est rempli en partant de la gauche, c'est à dire qu'e :*

- *Il n'existe aucun nœud de prodondeur $h-1$ qui n'a qu'un enfant droit*
- *Si un nœud de profondeur $h-1$ a seulement un enfant gauche, ou n'a aucun enfant, alors tous les nœuds situés à sa droite à la même profondeur n'ont aucun enfant.*

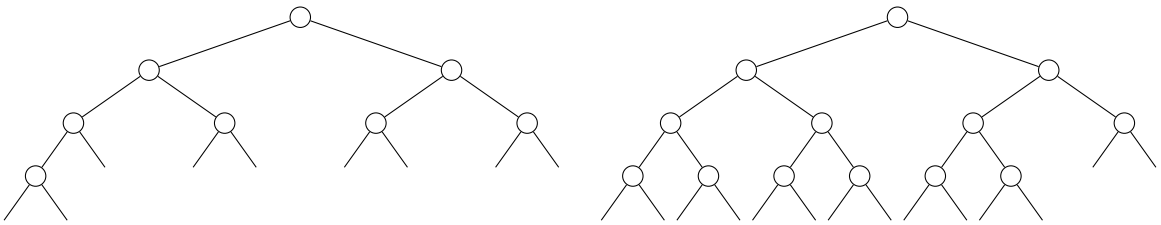


FIGURE 2 – Deux arbres binaires complets de hauteur 3.

Proposition 15. *La définition qu'on a donnée d'arbre parfait est équivalente à la fonçon dont on a utilisé la notion plus haut. Ent d'autres termes, les trois affirmations suivantes sont équivalentes pour tout arbre binaire A de hauteur $h \geqslant -1$:*

1. *A est parfait ;*
2. *A est de taille $2^{h+1} - 1$;*
3. *Si A est non vide, les deux sous-arbres immédiats de A sont parfaits de hauteur $h-1$.*

Un arbre parfait de hauteur h a 2^p nœuds à la profondeur p pour tout $p \in [\![0; h]\!]$.

1. Les nœuds de profondeur h n'ont évidemment aucun enfant, sinon ces enfants serait de profondeur $h+1$, qui est strictement supérieur à la hauteur.

Preuve Montrons l'équivalence par induction structurelle sur A :

- *Init :* Si A est l'arbre vide, les trois propositions sont trivialement vraies, donc équivalentes.
- *Hérédité :* Soit $A = \texttt{Noeud}(\_, A_g, A_d)$ un arbre binaire non vide de hauteur $h \geqslant 0$. On suppose l'équivalence vraie pour A_g et A_d.
  - $1 \implies 3$: Supposons A est parfait. Alors tous ses nœuds de profondeur comprise dans $[\![0; h-1]\!]$ ont exactement deux enfants. Or les nœuds de A_g et de A_d de profondeur p correspondent à des nœuds de A de profondeur $p+1$.

    Donc tous les nœuds de A_g et A_d de profondeur $[\![0; h-2]\!]$ ont exactement deux enfants. Comme A_g et A_d sont de hauteur au plus $h-1$, ils sont en fait de hauteur exactement $h-1$, donc ils sont parfaits, ce qui prouve la propriété 3.
  - $3 \implies 1$: Réciproquement, supposons que A a ses deux sous-arbres immédiats parfaits de hauteur $h-1$.

    Alors, par hypothèse d'induction, on a donc que chaque nœud de A_g (respectivement de A_d) de profondeur strictement inférieure à $h-1$ dans A_g (respectivement dans A_d) a exactement deux enfants.

    Or tout nœud de A de profondeur $p \in [\![1; h-1]\!]$ correspond à un nœud de A_g ou A_d de profondeur $p-1 \in [\![0; h-2]\!]$ dans cet arbre. Donc tout nœud de A de profondeur dans $[\![1; h-1]\!]$ a exactement deux enfants.

    De plus, $0 \in [\![0; h-1]\!]$ si et seulement si $h \geqslant 1$. Mais alors A_g et A_d sont de hauteur $h-1 \geqslant 0$, donc tous deux non vides, donc la racine de A a aussi deux enfants.

    Donc tout nœud de A de profondeur dans $[\![0; h-1]\!]$ a exactement deux enfants, ce qui est la propriété 1.
  - $3 \implies 2$:

    Supposons que A a ses deux sous-arbres immédiats parfaits de hauteur $h-1$.

    Alors, par hypothèse d'induction, on a donc que ces deux sous-arbres sont tous deux de taille $2^h - 1$.

    Donc A est de taille $2^h - 1 + 2^h - 1 + 1 = 2^{h+1} - 1$, ce qui correspond à la propriété 2.
  - $2 \implies 3$:

    Supposons A de taille $2^{h+1} - 1$. On sait que les deux sous-arbres de A sont de hauteur au plus $h-1$, donc tous deux de taille au plus $2^h - 1$.

    Or si $\text{taille}(A_g) < 2^h - 1$, alors $\text{taille}(A) = \text{taille}(A_g) + \text{taille}(A_d) < 2^h - 1 + 2^h - 1 + 1 = 2^{h+1} - 1$: contradiction.

    Donc A_g est de taille $2^h - 1$, et A_d de même par un argument symétrique.

    Donc par hypothèse d'induction, A_g et A_d sont parfaits de hauteur $h-1$.

Remarque 16. *Un arbre parfait est a fortiori complet.*

Un arbre complet de hauteur h a entre 2^h et $2^{h+1} - 1$ nœuds.

1.2 Parcours en profondeur d'un arbre binaire

Lorsqu'une fonction récursive s'appelle sur tous les nœuds d'un arbre, en faisant pour chaque nœud un appel récursif sur le SAG et le SAD, on parle de *parcours en profondeur*[1]. Un parcours en profondeur d'un arbre binaire est donc simplement un parcours qui exploite de la façon la plus simple la structure inductive des arbres.

Par convention, sauf indication explicite contraire, un parcours en profondeur est toujours de gauche à droite (c'est à dire qu'on effectue l'appel récursif sur le SAG avant celui sur le SAD).

1. En anglais : *Depth-First Search*, ou DFS. Le "first" fait référence au fait que l'arbre tout entier est parcouru, mais on va *d'abord* chercher les nœuds een profondeur. On parle parfois de *parcours en profondeur d'abord* en français.

Souvent, un algorithme qui effectue un parcours en profondeur va agir d'une certaine façon [2] sur les étiquettes de chaque nœud rencontré. Il y a trois moments où un appel de la fonction peut traiter l'étiquette du nœud en argument :

- entre le début de l'appel (la création de la frame), et le premier appel récursif sur le SAG ;
- Après l'appel récursif sur le SAG, quand la frame redevient active, avant le second appel récursif sur le SAD ;
- Après le second appel récursif, quand la frame redevient active à nouveau avant d'être détruite.

Si un parcours en profondeur ne traite l'étiquette des nœuds que pendant un seul de ces trois moments, on parle respectivement de parcours (en profondeur) *préfixe*, *infixe*, et *suffixe* [1].

Exemple 17. *On considère l'arbre de la figure 3. La séquences des étiquettes de l'arbre traitées par un parcours en profondeur est la suivante :*

- *Pour un parcours préfixe, on traite d'abord la racine d'étiquette* 42*, puis le sous-arbre gauche (en commençant évidemment par sa racine de la même façon), et enfin le sous-arbre droit. On a donc :*

$$42; 0; 1; 2; 3; 4; 5$$

- *Pour un parcours infixe, on traite d'abord toutes les étiquettes du sous-arbre gauche, puis la racine d'étiquette* 42*, et enfin les étiquettes du sous-arbre droit. On a donc :*

$$1; 0; 2; 42; 3; 5; 4$$

- *Pour un parcours suffixe, on traite d'abord toutes les étiquettes du sous-arbre gauche, puis du sous-arbre droit, et on ne traite l'étiquette de la racine qu'en dernier. On a donc :*

$$1; 2; 0; 5; 4; 3; 42$$

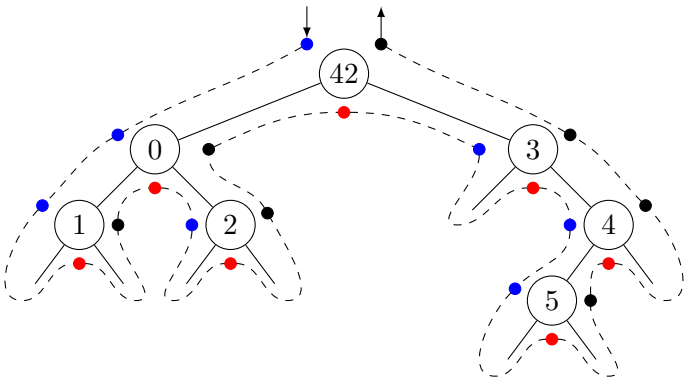


FIGURE 3 – Si on trace un chemin qui va de gauche à droite autour de l'arbre en commençant par la racine, on passe 3 fois à côté de chaque nœud : la première fois (•) correspond à un parcours préfixe, la deuxième (•) à un parcours infixe, et la troisième (•) à un parcours suffixe.

2. Par exemple : les modifier si elles sont mutables, les ajouter à un tableau associatif pour indiquer qu'elles ont été vues, vérifier qu'elles vérifient une certaine propriété... Le cas que nous traitons dans cette partie est celui où l'algo affiche les étiquettes lorsqu'il les traite.

1. On dit aussi *postfixe*.

1.3 Parcours en largeur d'un arbre binaire

Un *parcours en largeur*[1] d'un arbre consiste à ignorer la structure inductive et à parcourir l'arbre dans sa largeur, c'est à dire en commençant par la racine, puis tous les nœuds de profondeur 1, puis tous les nœuds de profondeur 2, et ainsi de suite.

Un parcours en largeur est plus courant sur un graphe que sur un arbre, puisqu'on préfère en général utiliser la structure inductive des arbres, mais on peut le définir sur un arbre. L'algorithme est le suivant :

```
Input : un arbre binaire A
Output : les étiquettes de A dans l'ordre d'un parcours en largeur
Algorithme :
  créer une file vide f
  si A non vide alors enfiler A dans f
  tantQue f non vide faire :
    défiler Noeud_Interne(x, A_g, A_d) de f
    afficher x
    si A_g non vide alors enfiler A_g dans f
    si A_d non vide alors enfiler A_d dans f
  finTantQue
```

1. En anglais : *Breadth-First Search*, ou BFS.

2 Arbres binaires stricts non vides

2.1 Définitions et propriétés

Définition 18. *Soient E et F des ensembles non vides. L'ensemble des arbres binaires stricts non vides (ABSNV) dont les nœuds internes sont étiquetés par E et les feuilles étiquetées par F est la structure inductive dont la signature est formée :*

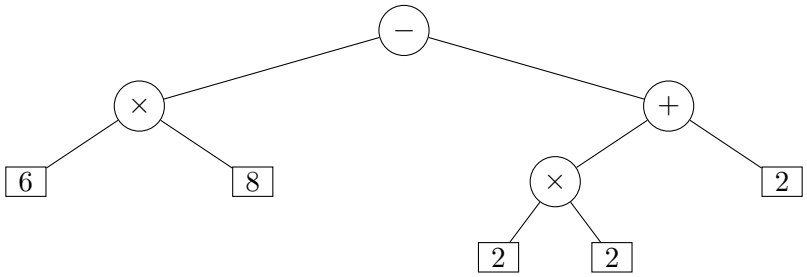
- *d'un constructeur constant, typé par F, noté* `Feuille` *;*
- *d'un constructeur d'arité 2, typé par E, noté* `Noeud_interne`*.*

Pour faire plus court, on parle aussi de l'ensemble des (E, F)-ABSNV.

En OCaml, on aura donc :

```
type ('a, 'b) absnv =
  Feuille of 'b
  | Noeud_interne of 'a * ('a, 'b) absnv * ('a, 'b) absnv
```

Exemple 19. *Si $E = \{+; -; \times; /\}$ et $F = \mathbb{N}$, on peut par exemple avoir l'ABSNV suivant qui représente une expression arithmétique :*



Remarque 20. *On peut évidemment avoir $E = F$ dans le cas où on utilise un ABSNV pour la forme d'arbre qu'il donne, plutôt que pour représenter une autre structure inductive comme dans l'exemple ci-dessus.*

Définition 21. *On a des définitions similaires pour les ABSNV à celles vues sur les arbres. Soit A un ABSNV :*

- *Les sous-arbres de A sont les ABSNV qui sont inférieurs ou égaux à A selon l'ordre structurel.*
- *Les nœuds de A sont les sous-arbres de A avec leur position dans l'écriture de A. Ils sont, par définition des ABSNV, tous non vides. On distingue les* nœuds internes *de A et les* feuilles *de A.*
- *La racine de A est le nœud qui correspond à A tout entier.*
- *L'étiquette d'un nœud est le paramètre (de E ou F selon si ce nœud est un nœud interne ou une feuille) qui correspond au constructeur de ce nœud.*
- *On garde les mêmes notions de SAG, SAD, parent, enfant, et profondeur.*
- *La taille de A est le nombre total de nœuds.*
- *La hauteur de A est définie récursivement de la même façon :*

$$\begin{cases} \forall y \in F, h(\texttt{Feuille}(y)) = 0 \\ \forall x \in E, A_g, A_d \text{ des } (E,F) - ABSNV, h(\texttt{Noeud\_Interne}(x, A_g, A_d)) = 1 + \max\Big(h(A_g); h(A_d)\Big) \end{cases}$$

Remarque 22. *On parle d'arbre binaire strict non vide car :*

- *chaque nœud a exactement 0 enfants (si c'est une feuille) ou 2 enfants (si c'est un nœud interne) : l'arbre est donc* strictement *binaire ;*
- *il n'y a pas d'arbre vide.*

Remarque 23. *Soit A un arbre binaire étiqueté par E de hauteur h.*
Alors on peut voir A comme un $(E, \{\texttt{Vide}\})$ de hauteur $h+1$.

Réciproquement, si A est un (E,F)-ABSNV de hauteur h, on peut aussi le voir comme un cas particulier d'un arbre binaire étiqueté par $E \cup F$, toujours de hauteur h.

Proposition 24. *Soit A un ABSNV, i son nombre de nœuds internes et f son nombre de feuilles.*
Alors $f = i+1$.
En particulier, la taille de A est $2i+1$ qui est impair.

Preuve On procède par induction structurelle sur A.

- *Init :* Si $A = \texttt{Feuille}(y)$ avec $y \in F$, alors on a bien $i = 0$ et $f = 1$.
- *Hérédité :* Soit $A = \texttt{Noeud\_Interne}(x, A_g, A_d)$, avec $x \in E$ et A_g, A_d des (E,F)-ABSNV. Notons i_g et i_d le nombre de nœuds internes de A_g et A_d respectivement, et f_g et f_d leurs nombres de feuilles. On suppose $f_g = i_g + 1$ et $f_d = i_d + 1$.
  Or toute feuille de A est une feuille de A_g ou de A_d, et les nœuds internes de A sont la racine de A et les nœuds internes de A_g et A_d.
  Donc $f = f_g + f_d$ et $i = i_g + i_d + 1$.
  Donc par hypothèse d'induction, $f = i_g + 1 + i_d + 1 = i + 1$.

Proposition 25. *Soit A un ABSNV de hauteur h et de taille n.*
Alors $2h+1 \leqslant n \leqslant 2^{h+1} - 1$.
De plus, pour tout $h \in \mathbb{N}$, il existe des ABSNV de hauteur h de tailles respetives $2h+1$ et $2^{h+1}-1$.

Preuve Il suffit de prouver la minoration de n, car la majoration est triviale en considérant un ABSNV comme un cas particulier d'arbre binaire. Montrons la par induction structurelle sur A.

- *Init :* Si $A = \texttt{Feuille}(y)$, $n = 1$ et $h = 0$. On a bien $2 \times 0 + 1 \leqslant 1$.
- *Hérédité :* Soit $A = \texttt{Noeud\_Interne}(x, A_g, A_d)$ un ABSNV non réduit à une feuille de hauteur $h \geqslant 0$ et de taille $n \geqslant 2$. Soient $h_g \geqslant 0$ et $h_d \geqslant 0$ les hauteurs de A_g et A_d, et n_g et n_d leurs tailles.
  Par définition, $h = 1 + \max(h_g; h_d)$. On a également remarqué que $n = 1 + n_g + n_d$.
  Or par hypothèse d'induction, on a $2h_g + 1 \leqslant n_g$ et $2h_d + 1 \leqslant n_d$.
  De plus comme h_g et h_d sont minorés par 0, $\max(h_g; h_d) \leqslant h_g + h_d$. C'est ici qu'on utilise le fait qu'on a des ABSNV et pas des arbres binaires généraux, pour prouver une minoration plus forte.
  Donc en particulier $2h + 1 = 2\max(h_g; h_d) + 3 \leqslant 2h_g + 1 + 2h_d + 1 + 1 \leqslant n_g + n_d + 1 = n$.

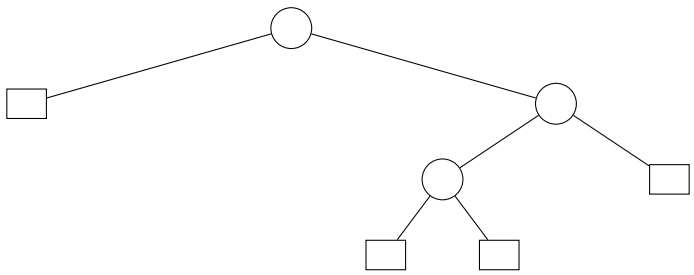


FIGURE 4 – Un peigne binaire strict non vide, de hauteur 3.

Remarque 26. *On a l'équivalent de la proposiotion 13 : la profondeur d'un nœud d'un ABSNV est bien définie, et est majorée par la hauteur de l'ABSNV.*

Ceci explique pourquoi le cas de base (un arbre réduit à une feuille) correspond à une hauteur nulle dans un ABSNV plutôt qu'une hauteur -1.

On peut définir les arbres parfaits parmi les ABSNV avec la profondeur, comme on l'a fait pour les arbres binaires : un ABSNV de hauteur $h \in \mathbb{N}$ est parfait si et seulement si tous ses nœuds de profondeur h sont des feuilles.

2.2 Parcours en profondeur préfixe d'un ABSNV

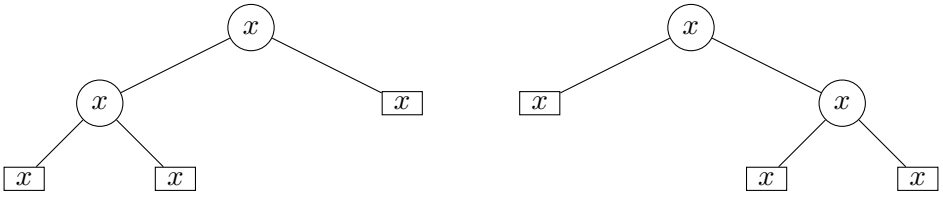
Proposition 27. *Soient E et F deux ensembles non vides distjoints, φ la fonction qui à un (E,F)-ABSNV associe la séquence des étiquettes de ses nœuds dans l'ordre préfixe.*

Alors φ est injective de l'ensemble des (E,F)-ABSNV dans $(E \cup F)^{(\mathbb{N})}$.

En particulier, un (E,F)-ABSNV est déterminé uniquement par l'écriture préfixe de ses étiquettes.

Remarque 28. *Si A est un ABNSV de taille n, alors $\varphi(A) \in (E \cup F)^n$.*

Remarque 29. *Il est très clair que si E et F ne sont pas disjoints, la proposition n'est plus vraie : en effet pour $x \in E \cap F$, il suffit de prendre les deux ABSNV distincts ci-dessous, dont l'écriture préfixe des étiquettes est simplement $(x;x;x;x;x)$.*

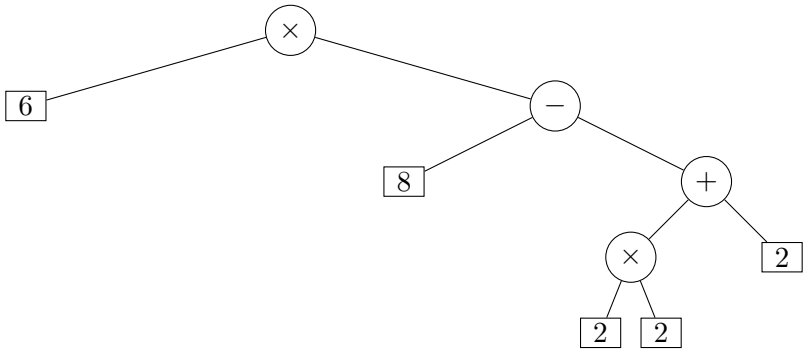


Remarque 30. *La même proposition est vraie pour l'écriture suffixe des étiquettes de l'arbre.*

En effet, l'écriture suffixe d'un ABSNV A est simplement le renversé de l'écriture préfixe de droite à gauche (c'est à dire, en faisant la récursion sur le SAD avant le SAG) de A.

Remarque 31. *La proposition est évidemment fausse pour l'écriture infixe. En effet, l'arbre ci-dessous et l'arbre de l'exemple 19 ont la même suite d'étiquettes dans l'ordre infixe :* $6 \times 8 - 2 \times 2 + 2$.

C'est pour cette raison qu'une expression arithmétique écrite en ordre infixe est usuellement parenthésée. On distingue ainsi $(6 \times 8) - (2 \times 2) + 2$, qui est l'expression correspondant à l'arbre de l'exemple 19, et $6 \times (8 - (2 \times 2) + 2)$, qui est l'expression correspondant à l'arbre ci-dessous.



Preuve On va montrer le lemme suivant, qui est plus fort que la proposition :

Lemme 32. *Soient E et F deux ensembles non vides distjoints, φ la fonction qui à un (E,F)-ABSNV associe la séquence des étiquettes de ses nœuds dans l'ordre préfixe.*

Soient A, B des (E,F)-ABSNV de tailles m et n tels que $\forall i \in [\![0; \min(m;n) - 1]\!], \varphi(A)[i] = \varphi(B)[i]$. Alors $A = B$.

En d'autres termes, le lemme affirme que si $\varphi(A)$ est un préfixe [1] de $\varphi(B)$, ou réciproquement, alors ils sont égaux et $A = B$. Ceci signifier qu'en plus d'être injective, l'image de φ n'admet pas deux tuples dont l'un est préfixe strict de l'autre [2].

1. Au sens où un mot est préfixe d'un autre. Ceci n'a pas de rapport direct avec le parcours préfixe de l'arbre qui définit φ.
2. Comme on le verra dans un contexte différent dans le chapitre sur le codage de Huffman, on peut dire que l'image de la fonction φ est un *code préfixe*.

Prouvons-le par induction structurelle sur A :

- *Init :* Soit $A = \texttt{Feuille}(y)$, et B qui vérifie l'hypothèse.
  $y \in F$ et donc $y \notin E$.
  On a donc $n = 1$, $\varphi(A) = (y)$, et donc comme B est un arbres strict non vide (donc $m \geqslant 1$), sa racine a pour étiquette $\varphi(B)[0] = y$.
  Comme y n'est pas un élément de E, nécessairement la racine de B est $\texttt{Feuille}(y)$, et donc $B = A$.
- *Hérédité :* Soit $A = \texttt{Noeud\_Interne}(x, A_g, A_d)$ un (E, F)-ABSNV de taille $n \geqslant 3$. On suppose que A_g et A_d vérifient la propriété.
  Soit B de taille m tel que $\forall i \in [\![0; \min(m; n) - 1]\!], \varphi(A)[i] = \varphi(B)[i]$.
  En particulier, $\varphi(B)[0] = x$, et $x \in E$ donc $x \notin F$.
  Donc la racine de B est de la forme $\texttt{Noeud\_Interne}(x, B_g, B_d)$.
  On a donc d'une part $\varphi(A)$ qui est constitué de x, suivi de $\varphi(A_g)$ et $\varphi(A_d)$, et d'autre part $\varphi(B)$ qui est constitué de x, suivi de $\varphi(B_g)$ et $\varphi(B_d)$.

  Soient n_g et m_g les tailles respectives de A_g et B_g. On a $1 \leqslant n_g \leqslant n - 2$ et $1 \leqslant m_g \leqslant m - 2$.
  De plus, $\begin{cases} \forall i \in [\![0; n_g - 1]\!], \varphi(A_g)[i] = \varphi(A)[i + 1] \\ \forall i \in [\![0; m_g - 1]\!], \varphi(B_g)[i] = \varphi(B)[i + 1] \end{cases}$.
  Donc $\forall i \in [\![0; \min(n_g; m_g) - 1]\!], \varphi(A_g)[i] = \varphi(B_g)[i]$ car $0 < i + 1 \leqslant \min(n; m) - 2 < \min(n; m) - 1$.
  Donc d'après l'hypothèse d'induction, on a $A_g = B_g$. En particulier, $m_g = n_g$.

  Soient $n_d = n - n_g - 1$ et $m_d = m - n_g - 1$ les tailles respectives de A_d et B_d.
  On a $\begin{cases} \forall i \in [\![0; n_d - 1]\!], \varphi(A_d)[i] = \varphi(A)[i + n_g + 1] \\ \forall i \in [\![0; m_d - 1]\!], \varphi(B_d)[i] = \varphi(B)[i + m_g + 1] = \varphi(B)[i + n_g + 1] \end{cases}$.
  On remarque que parce que $m_g = n_g$, les indices dans les termes de droite de ces deux égalités sont les mêmes.
  Donc $\forall i \in [\![0; \min(n_d; m_d) - 1]\!], \varphi(A_d)[i] = \varphi(B_d)[i]$ car $0 < i + n_g + 1 \leqslant \min(n; m) - 1$.
  Donc d'après l'hypothèse d'induction, on a $A_d = B_d$.
  On a donc que $B = \texttt{Noeud\_Interne}(x, A_g, A_d) = A$, ce qui conclut la preuve de l'hérédité et donc de la proposition

3 Arbres d'arité quelconque et forêts

Pour représenter une hiérarchie (par exemple, l'arborscence des dossiers et fichiers dans un système de fichiers, les processus en cours sur un ordinateur, des échelons administratifs....), il est pertinent d'utiliser un arbre où chaque nœud peut avoir un nombre arbitrairement grand d'enfants. On parle alors d'*arbres d'arité quelconque*, ou simplement *arbres* si le contexte ne présente pas d'ambiguïté.

Pour caractériser mathématiquement ces arbres, on utilise une autre structure inductive appelée *forêts*.

Définition 33. *Soit E un ensemble non vide. L'ensemble des forêts étiquetées par E et l'ensemble des arbres d'arité quelconque étiquetés par E dont deux structures mutuellement inductives :*

- *Un arbre d'arité quelconque est formé par un constructeur (que l'on note ici* `Node` *pour éviter la confusion) typé par un élément de E et appliqué à une forêt.*
- *Une forêt est une liste d'arbres d'arité quelconque. On peut donc la voir comme définie par un constructeur constant représentant la forêt vide, et un constructeur particulier d'arité 2 appliqué à un arbre et une forêt.*

En OCaml, on peut écrire :

```
type 'a arbre = Node of 'a * 'a foret
and 'a foret = 'a arbre list
```

Le mot-clé `and` ici indique deux définitions de type mutuellement inductives, comme lorsqu'on définit en une seule phrase deux fonctions mutuellement récursives avec `let rec .... and ....`

Remarque 34. *On peut aussi considérer qu'il n'y a qu'une seule structure inductive : les forêts. En effet une forêt est soit la forêt vide, soit un terme de la forme* `Node(`x`,` f_1`)::`f_2 *où x est un élément de E et f_1, f_2 sont des forêts.*

On pourait donc définir un type équivalent ainsi :

```
type 'a foret = Foret_vide | Foret_cons of 'a * 'a foret * 'a foret
```

Proposition 35. *Soit E un ensemble. Il existe une bijection canonique*[1] *φ entre l'ensemble des forêts étiquetées par E et l'ensemble des arbres binaires étiquetés par E, définie par :*

$$\begin{cases} \varphi(\texttt{[]}) = \texttt{Vide} \\ \forall x \in E, f_1, f_2 \text{ forêts}, \varphi(\texttt{Node}(x, f_1)\texttt{::}f_2) = \texttt{Noeud}(x, \varphi(f_1), \varphi(f_2)) \end{cases}$$

La preuve de cette proposition découle directement de la remarque. En effet, on peut similairement définir ψ la bijection réciproque et on vérifie aisément que $\varphi \circ \psi$ est l'identité sur les arbres binaires et $\psi \circ \varphi$ est l'identité sur les arbres d'arité quelconque.

Corollaire 36. *L'ensemble des arbres d'arités quelconques est en bijection avec l'ensemble des arbres binaires non vides dont le SAD est vide.*

Remarque 37. *Les définitions et théorèmes qui suivent sont exprimés du point de vue des arbres, même s'il peut être plus naturel de considérer la forêt d'arité quelconque en premier (puisque c'est la vraie structure inductive). La raison est que les arbres sont en général la structure qui nous intéresse vraiment pour ses applications pratiques, plutôt que les forêts.*

Définition 38. *On a des définitions similaires pour les arbres d'arités quelconques à celles vues sur les arbres. Soit A un arbre d'arité quelconque, f une forêt :*

- *Les sous-forêts de f sont les forêts qui sont inférieures ou égales à f selon l'ordre structurel.*
- *Les sous-arbres de A sont les abres qui sont inférieurs ou égaux à A selon l'ordre structurel, c'est à dire les arbres qui sont un élément d'une sous-forêt de $[A]$.*
- *Les nœuds de A sont les sous-arbres de A avec leur position dans l'écriture de A. Les nœuds de f sont les sous-arbres des éléments de f.*
- *La racine de A est le nœud qui correspond à A tout entier.*
- *L'étiquette d'un nœud est le paramètre de E qui correspond au constructeur de ce nœud.*
- *Si $N =$* `Node`(x, f_1)*, les nœuds qui composent f_1 sont appelés les enfants de N, et si f_1 est non vide, sa tête est appelée l'*enfant le plus à gauche[2] *de N. Tout nœud de f_1 a pour parent N.*
  De plus, dans la forêt $N\texttt{::}M\texttt{::}f_2$ de taille au moins 2, M est appelé le frère droit[3] *de N.*

Remarque 39. *La bijection explicitée plus haut envoie donc (s'ils existent) l'enfant le plus à gauche d'un nœud N sur l'enfant gauche de $\varphi(N)$ et le frère droit de N sur l'enfant droit de $\varphi(N)$.*

1. La bijection est canonique, c'est à dire que tout le monde est d'accord que c'est la bonne, par contre le nom φ n'a rien de canonique. Si vous l'utilisez dans une copie, précisez de quoi il s'agit.
2. En anglais : *leftmost child.*
3. En anglais : *right sibling.* Ici je n'ai pas gardé la terminologie non genrée car je trouve que "adelphe droit" ça sonne bizarrement.

Définition 40. *On définit trois paramètres sur les arbres d'arité quelconque et sur les forêts.*

- *La taille d'un arbre (respectivement d'une forêt) est son nombre de nœuds.*
- *La hauteur est définie ainsi :*
  - *Pour tout arbre* $A = \texttt{Node}(x, f)$, $\text{hauteur}(A) = 1 + \max\limits_{A' \text{ arbre de } f}\Big(\text{hauteur}(A')\Big)$ *si* f *est non vide, et* 0 *sinon.*
  - *Pour toute forêt* f*, la hauteur de* f *est le maximum des hauteurs des arbres de* f*. si* f *est non vide, et* -1 *sinon.*
- *L'*arité maximale*, ou simplement* arité *s'il n'y a pas de risque de confusion, est définie ainsi :*
  - *Pour toute forêt* f*, l'arité maximale de* f *est le maximum des longueurs (au sens usuel de la longueur d'une liste) des sous-forêts de* f*.*
  - *Pour tout arbre* $A = \texttt{Node}(x, f)$*, l'arité maximale de* A *est l'arité maximale de* f*.*

Proposition 41. *Les définitions précédentes peuvent donner une caractérisation inductive sur les forêts utilisant uniquement les deux sous-forêts immédiates d'une forêt non vide. Notons* n*,* h*,* a *et* ℓ *les fonctions taille, hauteur, arité maximale et longueur sur les forêts.*

Alors :

- $n(\texttt{[]}) = 0$*,* $h(\texttt{[]}) = -1$*, et* $a(\texttt{[]}) = 0$*.*
- *Si* f *est une forêt non vide, alors* $f = \texttt{Node}(x, f_1)\texttt{::}f_2$*. Dans ce cas, on a :*

$$n(f) = 1 + n(f_1) + n(f_2)\ , \quad h(f) = \max\Big(1 + h(f_1); h(f_2)\Big) \quad et \quad a(f) = \max\Big(a(f_1); a(f_2); 1 + \ell(f_2)\Big)$$

De plus, pour tout $A = \texttt{Node}(x, f)$*,* $\text{taille}(A) = 1 + n(f)$*,* $\text{hauteur}(A) = 1 + h(f)$ *et* $\text{arite}(A) = a(f)$*.*

Preuve Les cas de bases sont tous par définition. On se place donc dans le cas $f = \texttt{Node}(x, f_1)\texttt{::}f_2$. Prouvons les trois relations de récurrence (celle sur la longueur des listes étant déjà connue) :

- Par définition, $n(f)$ est le nombre de nœuds de f, or tout noœud est soit $\texttt{Node}(x, f_1)$ la tête de f, soit un sous-arbre de la tête de f (et donc un nœud de f_1), soit un nœud de f_2 la queue de f.
  Donc on a bien $n(f) = 1 + n(f_1) + n(f_2)$.
- Par définition, de la hauteur, on a :

$$\begin{aligned} h(f) &= \max_{A \text{ élément de } f}\Big(\text{hauteur}(A)\Big) \\ &= \max\left(\text{hauteur}\left(\texttt{Noeud}(x, f_1)\right); \max_{A \text{ élément de } f_2}\Big(\text{hauteur}(A)\Big)\right) \\ &= \max\left(1 + \max_{A \text{ élément de } f_1}\Big(\text{hauteur}(A)\Big); h(f_2)\right) \\ &= \max\Big(1 + h(f_1); h(f_2)\Big) \end{aligned}$$

- Toute sous-forêt de f est soit f elle-même, soit une sous forêt de f_1, soit une sous-forêt de f_2. On a donc :

$$\begin{aligned} a(f) &= \max_{f' \text{sous-forêt de } f}\Big(\ell(f')\Big) \\ &= \max\left(\max_{f' \text{sous-forêt de } f_1}\Big(\ell(f')\Big), \max_{f' \text{sous-forêt de } f_2}\Big(\ell(f')\Big), \ell(f)\right) \\ &= \max\Big(a(f_1), a(f_2), \ell(f)\Big) \\ &= \max\Big(a(f_1), a(f_2), 1 + \ell(f_2)\Big) \qquad \text{par définition de la longueur d'une liste} \end{aligned}$$

Proposition 42. *Soit A un arbre d'arité quelconque, et B l'arbre binaire non vide de SAD vide qui lui correspond d'après la bijection canonique.*

Alors $\text{taille}(A) = \text{taille}(B)$ *et* $\text{hauteur}(A) \leqslant \text{hauteur}(B) \leqslant \text{arite}(A) \times \text{hauteur}(A)$.

De plus, pour tout $h_0 \in \mathbb{N}$, il existe A_1 et A_2 arbres d'arité quelconques tous deux de hauteur h_0, B_1 et B_2 les arbres binaires non vides de SAD vide qui leur correspondent, tels que $\text{hauteur}(B_1) = h_0$ *et* $\text{hauteur}(B_2) = h_0 \times \text{arite}(A_2)$.

Preuve Notons N et H les fonctions tailles et hauteur sur les arbres binaires.

On sait que, si φ est la bijection décrite plus haut entre l'ensemble des forêts et l'ensemble des arbres binaires, il suffit de montrer pour toute forêt f que
$n(f) = N(\varphi(f))$ et $h(f) \leqslant H(\varphi(f)) \leqslant h(f)a(f)$

En effet, $B = \varphi([A])$, et on a :

- $n(A) = \text{taille}(A)$;
- $h(A) = \text{hauteur}(A)$;
- $a([A]) = \max\Big(1; \text{arite}(A)\Big)$, or seul un arbre réduit à un nœud (donc de hauteur nulle) peut avoir une arité nulle, et comme $h([A]) = \text{hauteur}(A) \geqslant 0$, on a :
  $h([A]) \times a([A]) = \max\Big(0 \times 1; \text{hauteur}(A) \times \text{arite}(A)\Big) = \text{hauteur}(A) \times \text{arite}(A)$

On va montrer par induction structurelle sur f la propriété suivante, dont découlera immédiatement la proposition voulue (car $l(f) \leqslant a(f)$) :

$$\forall f \text{ forêt}, n(f) = N(\varphi(f)) \text{ et } h(f) \leqslant H(\varphi(f)) \leqslant (h(f)-1)a(f) + l(f)$$

- *Init :* On considère la forêt vide $f =$ `[]`. Alors $n(f) = 0$, $h(f) = -1$, $a(f) = 0$ et $l(f) = 0$.
  On a $\varphi(f) =$ `Vide`, donc $N(\varphi(f)) = 0$ et $H(\varphi(f)) = -1$.
  On a bien $0 = 0$ et $-1 \leqslant -1 \leqslant (-1-1) \times 0 + 0$.
- *Hérédité :* Soit $f =$ `Node`(x, f_1)`::`f_2 une forêt non vide.
  On suppose la propriété vraie pour f_1 et f_2.
  Par définition, on a $\varphi(f) =$ `Noeud`$(x, \varphi(f_1), \varphi(f_2))$.
  On a donc :

$$\begin{aligned} N(\varphi(f)) &= 1 + N(\varphi(f_1)) + N(\varphi(f_2)) \\ &= 1 + n(f_1) + n(f_2) \qquad \text{par HI} \\ &= n(f) \qquad \text{d'après la proposition 41} \end{aligned}$$

  De même :

$$\begin{aligned} H(\varphi(f)) &= 1 + \max\Big(H(\varphi(f_1)); H(\varphi(f_2))\Big) \\ &\geqslant 1 + \max\Big(h(f_1); h(f_2)\Big) \qquad \text{par HI} \\ &\geqslant \max\Big(1 + h(f_1); h(f_2)\Big) \\ &\geqslant h(f) \qquad \text{d'après la proposition 41} \end{aligned}$$

Enfin :

$$\begin{aligned}
H(\varphi(f)) &= 1 + \max\Big(H(\varphi(f_1)); H(\varphi(f_2))\Big) \\
&\leqslant 1 + \max\Big((h(f_1)-1)a(f_1) + l(f_1); ((h(f_2)-1)a(f_2) + l(f_2)\Big) && \text{par HI} \\
&\leqslant 1 + \max\Big((h(f_1)-1)a(f) + a(f); (h(f_2)-1)a(f) + (l(f)-1)\Big) \\
&\text{en majorant } l(f_1) \leqslant a(f_1) \text{ et } a(f_2) \text{ par } a(f) \\
&\leqslant 1 + \max\Big(h(f_1)a(f); (h(f_2)-1)a(f) + l(f) - 1\Big) \\
&\leqslant \max\Big(h(f_1)a(f) + 1; (h(f_2)-1)a(f) + l(f)\Big) \\
&\leqslant \max\Big((h(f_1)+1-1)a(f) + l(f); (h(f_2)-1)a(f) + l(f)\Big) && \text{car } l(f) \geqslant 1 \\
&\leqslant \Big(\max\Big(h(f_1)+1; h(f_2)\Big) - 1\Big)a(f) + l(f) \\
&\leqslant (h(f)-1)a(f) + l(f) && \text{d'après la proposition 41}
\end{aligned}$$