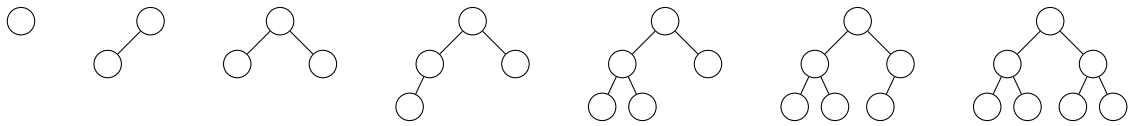
Outils logiques et algorithmiques – TD 9 – Correction

Exercice 1

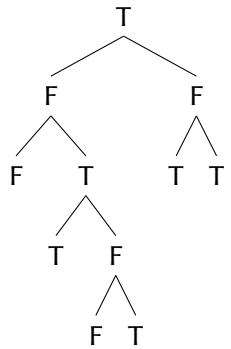
1. Hauteur 0 : uniquement l'arbre vide. Hauteur 1 : uniquement l'arbre ci-dessous à gauche. Hauteur 2 : les deux suivants. Hauteur 3 : les quatre derniers.



2. Le nombre d'arbres quasi-parfait de hauteur k est égal au nombre de feuilles d'un arbre quasi-parfait de hauteur k, c'est-à-dire 2^{k-1}.
3. Minimum : 2^{k-1} (les $2^{k-1} - 1$ nœuds internes d'un arbre parfait, plus sa feuille la plus à gauche). Maximum : $2^k - 1$ (cas de l'arbre parfait).
4. Au minimum 0 (arbre parfait). Au maximum 1 (le parent de la dernière feuille, si celle-ci est à sa gauche).
5. Au minimum 2^{k-2} si $k > 1$ (tous les nœuds à profondeur $k-2$ sont des feuilles sauf celui le plus à gauche qui a un unique fils, où ce fils est lui-même une feuille), au maximum 2^{k-1}.

Exercice 2

1. Le mot vide ϵ correspond à la racine de l'arbre, qui est donc étiquetée par T ici. Les mots 01, 010 et 0111, commençant par 0, sont ensuite donnés par le sous-arbre gauche, et les mots 10 et 11, commençant par 1, sont donnés par le sous-arbre droit. Comme dans l'exemple de l'énoncé, dans le dessin ci-dessous on ne fait apparaître que les nœuds, et pas les symboles E.



Notez que les nœuds étiquetés par F et qui n'ont pas de fils auraient pu être remplacés par E, et donc auraient pu ne pas apparaître.

2. Pour commencer, il faut prévoir au moins une équation pour le cas E et une pour le cas N. Dans le cas E nous avons un arbre vide, qui ne représente un ensemble vide de mots : le cardinal est zéro. Dans le cas N(g, b, d), il faut additionner les cardinaux des ensembles correspondant aux deux sous-arbres g et d, mais aussi compter un mot si l'étiquette b est T. Pour représenter cette dernière condition on donne deux équations pour le cas N, selon que l'étiquette vaut T ou F.

$$\begin{aligned} \text{cardinal}(\mathsf{E}) &= 0 \\ \text{cardinal}(\mathsf{N}(g, \mathsf{F}, d)) &= \text{cardinal}(g) + \text{cardinal}(d) \\ \text{cardinal}(\mathsf{N}(g, \mathsf{T}, d)) &= 1 + \text{cardinal}(g) + \text{cardinal}(d) \end{aligned}$$

3. Le principe de récurrence énoncé qu'une propriété est vraie pour tous les arbres dès lors qu'elle vaut pour le cas de base (E) et dès lors qu'elle est préservée par la combinaison de deux arbres avec N. On peut dont l'énoncer comme suit.

   Pour toute propriété P, si

   — $P(\mathsf{E})$, et

— $P(\mathsf{N}(g, b, d))$ pour tous g et d tels que $P(g)$ et $P(d)$,

alors pour tout arbre préfixe p, $P(p)$.

4. Pour une preuve par récurrence, il faut appliquer le principe précédent en prenant une propriété P concrète, en général correspondant directement à notre objectif. Ici, la propriété $P(p)$ choisie est donc « $\mathsf{cardinal}(p) \leq \mathsf{nbNœuds}(p)$ ».

   La preuve par récurrence consiste ensuite en la vérification des deux conditions.

   — Cas de base : il faut vérifier que notre propriété P est vraie pour le cas de base E, autrement dit montrer que $\mathsf{cardinal}(F) \leq \mathsf{nbNœuds}(\mathsf{E})$.

   C'est immédiat en explicitant les valeurs de cardinal et nbNœuds sur cet arbre. $\mathsf{cardinal}(F) = 0 \leq 0 = \mathsf{nbNœuds}(\mathsf{E})$

   — Cas inductifs : en partant de deux sous-arbres g et d pour lesquels la propriété P est vraie, il faut vérifier qu'elle l'est encore pour l'arbre combiné $\mathsf{N}(g, b, d)$.

   On suppose donc $\mathsf{cardinal}(g) \leq \mathsf{nbNœuds}(g)$ et $\mathsf{cardinal}(d) \leq \mathsf{nbNœuds}(d)$ pour des arbres g et d arbitraires (c'est-à-dire qu'on ne sait rien d'autre sur g et d), et on tente de démontrer $\mathsf{cardinal}(\mathsf{N}(g, b, d)) \leq \mathsf{nbNœuds}(\mathsf{N}(g, b, d))$.

   Pour cela on remarque que, par définition, $\mathsf{cardinal}(\mathsf{N}(g, b, d))$ vaut soit $\mathsf{cardinal}(g) + \mathsf{cardinal}(d)$ soit $1+\mathsf{cardinal}(g)+\mathsf{cardinal}(d)$ (selon la valeur de b), et donc que dans tous les cas $\mathsf{cardinal}(\mathsf{N}(g, b, d)) \leq \mathsf{cardinal}(g) + \mathsf{cardinal}(d)$. On peut ensuite utiliser les hypothèses comparant cardinal et nbNœuds pour g et d, puis conclure avec la définition de nbNœuds. En résumé :

$$
\begin{array}{lll}
& \mathsf{cardinal}(\mathsf{N}(g, b, d)) & \\
\leq & 1 + \mathsf{cardinal}(g) + \mathsf{cardinal}(d) & \\
\leq & 1 + \mathsf{nbNœuds}(g) + \mathsf{nbNœuds}(g) & \quad IH \\
= & \mathsf{nbNœuds}(\mathsf{N}(g, b, d)) &
\end{array}
$$

5. Concernant la forme de ce raisonnement, on peut déjà remarquer qu'il ne s'intéresse qu'au cas inductif, et ne présente pas le cas de base de la récurrence. Il aurait fallu commencer par vérifier que la propriété était vraie pour l'arbre E pour obtenir une preuve complète.

   En outre, on peut remarquer que la propriété est en réalité fausse. Par exemple en prenant le cas de l'arbre $\mathsf{N}(\mathsf{E}, \mathsf{T}, \mathsf{E})$, qui représente l'ensemble $\{\varepsilon\}$ contenant le mot vide, on a un cardinal de 1 et une somme de 0. De même, en considérant le cas de base E (justement celui qui a été oublié dans la preuve !) on aurait eu un cardinal et une somme de 0, ce qui donne encore un contre-exemple.

6. On obtient un type Caml en reprenant directement les symboles E et N proposés par l'énoncé et en associant chacun aux types des arguments auxquels il s'applique.
   `type prf = E | N of prf * bool * prf`

7. Pour la fonction `appartient` on prévoit trois cas :

   — d'abord, éliminer le cas de l'arbre vide E, qui ne contient aucun mot,

   — ensuite, on peut regarder le mot cherché

      — s'il est vide, on consulte l'étiquette du nœud courant

      — sinon, on cherche dans le sous-arbre gauche ou droit, en fonction de la première lettre

```
let rec appartient m p = match (m, p) with
  | _, E -> false
  | [], N(g, b, d) -> b
  | b::m, N(g, _, d) ->
    if b then appartient m d else appartient m g
```

8. La fonction `ajoute` est séparée de même en trois cas :

   — le cas de l'ajout dans un arbre vide E est traité à part (fonction auxiliaire commentée ci-dessous)

— dans le cas d'un arbre non vide on consulte le mot
  — s'il est vide, on fait passer l'étiquette du nœud courant à `true` (plus précisément, on crée un nouveau nœud avec les mêmes sous-arbres et cette nouvelle étiquette)
  — sinon, on fait l'insertion dans le sous-arbre gauche ou droit en fonction de la première lettre.

La fonction auxiliaire d'ajout dans l'arbre vide ne consulte que le mot `m` à ajouter, et renvoie un arbre contenant uniquement ce mot.

```
let rec ajoute_vide m = match m with
  | [] -> N(E, true, E)
  | b::m ->
    let p = ajoute_vide m in
    if b then N(E, false, p) else N(p, false, E)

let rec ajoute m p = match (m, p) with
  | _, E -> ajoute_vide m
  | [], N(g, b, d) -> N(g, true d)
  | b::m, N(g, b', d) ->
    if b then N(g, b', ajoute m d) else N(ajoute m g, b', d)
```

Exercice 3

1. Un cas pour chaque forme d'abre

$$\begin{aligned} \mathsf{in}(n, \mathsf{E}) &= \text{faux} \\ \mathsf{in}(n, \mathsf{N}(a_1, m, a_2)) &= n = m \| \mathsf{in}(n, a_1) \| \mathsf{in}(n, a_2) \end{aligned}$$

2. Un cas pour chaque forme d'arbre

$$\begin{aligned} \mathsf{infix}(\mathsf{E}) &= \epsilon \\ \mathsf{infix}(\mathsf{N}(a_1, n, a_2)) &= \mathsf{infix}(a_1) \cdot n \cdot \mathsf{infix}(a_2) \end{aligned}$$

3. Le singleton $\mathsf{N}(\mathsf{E}, n, \mathsf{E})$ est un arbre binaire de recherche ainsi que le premier arbre $\mathsf{N}(\mathsf{S}(1), 2, \mathsf{S}(3))$. Le dernier arbre $\mathsf{N}(\mathsf{E}, 2, \mathsf{N}(\mathsf{S}(1), 3, \mathsf{E}))$ n'est pas un arbre binaire de recherche car le sous-arbre à droite de 2 contient 1.
4. Exemples : $\mathsf{N}(\mathsf{N}(\mathsf{S}(1), 2, \mathsf{S}(3)), 4, \mathsf{E})$ et $\mathsf{N}(\mathsf{S}(1), 2, \mathsf{N}(\mathsf{S}(3), 4, \mathsf{E}))$. Dans tous les cas la fonction infix donne 1234.
5. Par récurrence sur a, on démontre $P(a)$: « si $a \in \mathcal{A}_r$ alors $\mathsf{infix}(a)$ est triée ».
   — Cas de base : arbre vide E, qui donne la suite vide.
   — Cas inductif : $a = \mathsf{N}(a_1, n, a_2)$, avec a_1 et a_2 vérifiant P. Supposons $a \in \mathcal{A}_r$. On a donc :
     — $a_1 \in \mathcal{A}_r$ et $a_2 \in \mathcal{A}_r$
     — pour tout élément k dans a_1, $k \leq n$
     — pour tout élément k dans a_2, $n \leq k$

   Par définition $\mathsf{infix}(a) = \mathsf{infix}(a_1) \cdot n \cdot \mathsf{infix}(a_2)$. On vérifie que deux éléments consécutifs de $\mathsf{infix}(a)$ sont par ordre croissants :
     — si tous deux dans $\mathsf{infix}(a_1)$, ok par hypothèse de récurrence sur a_1
     — si tous deux dans $\mathsf{infix}(a_2)$, ok de même
     — si dernier de $\mathsf{infix}(a_1)$ et n, ok par définition ABR
     — si n et premier de $\mathsf{infix}(a_2)$, ok par définition ABR

6. On cherche à gauche (resp. à droite) lorsque $n < x$ (resp. $x < n$). Pas besoin de chercher à gauche ni à droite si $n = x$.
7. On sépare le cas node en trois pour tenir compte des critères de la question précédente.

$$\begin{aligned}
\mathsf{inabr}(n, \mathsf{E}) &= \mathsf{faux} \\
\mathsf{inabr}(n, \mathsf{N}(a_1, n, a_2)) &= \mathsf{vrai} \\
\mathsf{inabr}(n, \mathsf{N}(a_1, x, a_2)) &= \mathsf{inabr}(n, a_1) \qquad \text{si } n < x \\
\mathsf{inabr}(n, \mathsf{N}(a_1, x, a_2)) &= \mathsf{inabr}(n, a_2) \qquad \text{si } x < n
\end{aligned}$$

8. Dans la récurrence sur a, on ajoute un raisonnement par cas pour s'intéresser aux manières dont $\mathsf{inabr}(n, a)$ peut valoir vrai (inversion).
9. Il faut prendre un arbre qui n'est pas trié. Par exemple chercher 5 dans 

10. Dans la récurrence sur a, on ajoute un raisonnement par cas sur les manières dont $\mathsf{in}(n, a)$ peut être vérifiée (inversion) et on élimine les cas incompatibles avec les hypothèses de $a \in \mathcal{A}_r$.
11. Lorsque l'on vérifie les deux sous-arbres d'un nœud, on met à jour les bornes en se servant de la valeur de ce nœud.

$$\begin{aligned}
\mathsf{verif}(min, max, \mathsf{E}) &= \mathsf{true} \\
\mathsf{verif}(min, max, \mathsf{N}(a_1, n, a_2)) &= min \leq n \leq max \;\&\&\; \mathsf{verif}(min, n, a_1) \;\&\&\; \mathsf{verif}(n, max, a_2)
\end{aligned}$$

12. Les bornes min et max doivent être respectivement un minorant et un majorant de l'ensemble des éléments. Cela fonctionne à coup sûr si on s'autorise $-\infty$ et $+\infty$. De manière plus subtile on peut choisir les éléments à l'extrême gauche et à l'extrême droite de l'arbre.
13. Récurrence sur a, avec inversion de la condition $\mathsf{verif}(m_1, m_2, a) = \mathsf{vrai}$.