Corrigé : automates d'arbre (Mines 2015)

Partie I. Fonctions utilitaires

Question 1. Il s'agit de redéfinir la fonction `(fun li x -> mem x li)` :

```
let rec contient li x = match li with
  | []   -> false
  | t::q -> x = t || contient q x ;;
```

La fonction **`contient`** est de complexité $O(|l_i|)$.

Question 2. Il s'agit maintenant de redéfinir la fonction **`union`** :

```
let rec union l1 l2 = match l1 with
  | []                      -> l2
  | t::q when contient l2 t -> union q l2
  | t::q                    -> t::(union q l2) ;;
```

La fonction **`contient`** est appelée $|l_1|$ fois lors du calcul de l'union de deux listes l_1 et l_2, donc la complexité de la fonction **`union`** est en $O(|l_1| \times |l_2|)$.

Question 3. On définit :

```
let rec fusion l = it_list union [] l ;;
```

La complexité $C(k)$ de la fonction **`fusion`** appliquée à $l = (l_1, \dots, l_k)$ vérifie la relation de récurrence :

$$C(k) = C(k-1) + O\Big(\big(|l_1| + |l_2| + \cdots + |l_{k-1}|\big) \times |l_k|\Big).$$

On en déduit que $C(k) = O\Big(\sum_{i=1}^{k} |l_i| \times \big(\sum_{j=1}^{i-1} |l_j|\big)\Big)$, quantité que l'on peut majorer par un $O(L^2)$ avec $L = |l_1| + \cdots + |l_k|$.

Question 4. On définit enfin le produit cartésien de deux listes :

```
let rec produit l1 l2 = match l1 with
  | []   -> []
  | t::q -> (map (function x -> (t, x)) l2) @ (produit q l2) ;;
```

Le coût de la concaténation est linéaire vis-vis-vis de son premier argument, donc la complexité de la fonction **`produit`** vérifie la relation : $C(|l_1|, |l_2|) = C(|l_1| - 1, |l_2|) + O(|l_2|)$. Sachant que $C(0, |l_2|) = O(1)$ on en déduit que $C(|l_1|, |l_2|) = O(|l_1| \times |l_2|)$.

Partie II. Arbres binaires étiquetés

Question 5.

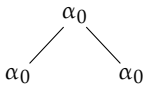
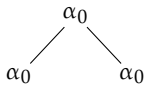
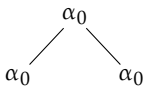
```
let arbre x ag ad = Noeud {etiquette = x; gauche = ag; droit = ad} ;;
```

Question 6.

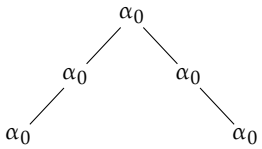
```
let rec taille_arbre = function
  | Vide    -> 0
  | Noeud n -> 1 + taille_arbre n.gauche + taille_arbre n.droit ;;
```

Partie III. Langages d'arbres

Question 7.

	appartient au langage	n'appartient pas au langage
L_0		
$L_{complet}$		
$L_{chaîne}$		
$L_{impartial}$		

Question 8. Un arbre complet est caractérisé par l'égalité : $S_g = S_d$, un arbre impartial par l'égalité : $|S_g| = |S_d|$. Tout arbre complet est donc impartial, mais la réciproque est fausse, comme le montre l'arbre ci-dessous, impartial mais non complet :



Question 9. Par hypothèse, pour tout $u \in S \setminus \{r\}$, $|g^{-1}(\{u\})| + |d^{-1}(\{u\})| = 1$ donc $S = \{r\} \cup g(S_g) \cup d(S_d)$, cette union étant disjointe.
Puisque g et d sont injectives, $|S| = 1 + |S_g| + |S_d|$ et puisque l'arbre est impartial, $|S_g| = |S_d|$, ce qui montre que $|S|$ est impair.

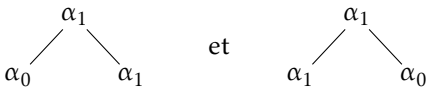
Partie IV. Automates d'arbres descendants déterministes

Question 10. Posons $Q = \{q_0, q_1, q_2\}$, $F = \{q_0, q_1\}$ et définissons δ par : $\forall c \in \Sigma$, $\delta(q_0, c) = (q_0, q_1)$, $\delta(q_1, c) = (q_2, q_2)$, $\delta(q_2, c) = (q_2, q_2)$.
Montrons que cet automate $\mathcal{A}^{\downarrow}$ reconnaît $L_{chaîne}$, en commençant par considérer un arbre-chaîne t : on a $S_d = \emptyset$. Si $t = \varepsilon$, t est reconnu puisque $q_0 \in F$. Sinon, considérons l'application $\varphi : S \to Q$ définie par $\forall u \in S$, $\varphi(u) = q_0$.

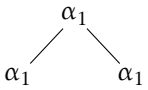
- Pour tout $u \in S$, on a $\delta(\varphi(u), \lambda(u)) = \delta(q_0, \lambda(u)) = (q_0, q_1)$.
- Si $u \in S_g$, on a bien $\varphi(g(u)) = q_0$, si $u \notin S_g$, on a bien $q_0 \in F$.
- Et dans tous les cas, $u \notin S_d$ et $q_1 \in F$.

Réciproquement, si t n'est pas un arbre-chaine, il existe un sommet u tel que $u \in S_d$. Dans ce cas, s'il existait une fonction φ vérifiant les conditions requise pour que $\mathcal{A}^{\downarrow}$ reconnaisse t, on aurait $\varphi(d(u)) = q_1$. Or $\delta(q_1, \lambda(d(u))) = (q_2, q_2)$ et $q_2 \notin F$: $d(u)$ doit nécessairement avoir un fils (gauche ou droite). Considérons alors un descendant v de $d(u)$ n'ayant pas de fils (il en existe forcément). Puisque q_2 est un puits, on a $\varphi(v) = q_2$, et puisque que $q_2 \notin F$, ceci contredit les propriétés de φ.

Question 11. Considérons un automate descendant déterministe $\mathcal{A}^{\downarrow} = (Q, q_0, F, \delta)$ reconnaissant L_0. Il doit en particulier reconnaître les arbres suivants :



Soit $(q_1, q_2) = \delta(q_0, \alpha_1)$.
Posons $(q_3, q_4) = \delta(q_2, \alpha_1)$. Puisque l'arbre de gauche est reconnu on doit avoir $q_3 \in F$ et $q_4 \in F$.
Posons $(q_5, q_6) = \delta(q_1, \alpha_1)$. Puisque l'arbre de droite est reconnu on doit avoir $q_5 \in F$ et $q_6 \in F$.
Mais alors $\mathcal{A}^{\downarrow}$ reconnaît l'automate suivant, ce qui est absurde :



Question 12.

```
let rec applique_desc add q = function
  | Vide    -> add.finals_desc.(q)
  | Noeud t -> let qg, qd = add.transitions_desc.(q).(t.etiquette) in
               applique_desc add qg t.gauche && applique_desc add qd t.droit ;;
```

Question 13.

```
let evalue_desc add t = applique_desc add 0 t ;;
```

Partie V. Automates descendants et langages rationnels de mots

Question 14. Soit x un mot de L, et $t = \text{chaîne}(x)$. Si $x = \varepsilon$ alors $q_0 \in F$ car $\mathcal{A}$ reconnaît x, et $t = \varepsilon$ est bien reconnu par $\mathcal{A}^{\downarrow}$.
Si $x = x_1 \cdots x_l$, il existe un chemin $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \cdots \xrightarrow{x_l} q_l$ dans $\mathcal{A}$ tel que $q_l \in F$.
Posons $t = (\{u_1, \ldots, u_l\}, u_1, \lambda, g, d)$ et définissons la fonction $\varphi : u_i \mapsto q_{i-1}$.

- on a bien $\varphi(u_1) = q_0$;
- pour tout $i \in [\![1, l-1]\!]$, $\delta'(\varphi(u_i), x_i) = (q_i, q_1')$. $g(u_i)$ est défini et $\varphi(g(u_i)) = \varphi(u_{i+1}) = q_i$, $d(u_i)$ n'est pas défini mais $q_1' \in F \cup \{q_1'\}$.
- pour $i = l$, $\delta'(\varphi(u_l), x_l) = (q_l, q_1')$. Ni $g(u_l)$ ni $d(u_l)$ ne sont définis, mais $q_l \in F$ et $q_1' \in F \cup \{q_1'\}$.

De ceci il résulte que $\mathcal{A}^{\downarrow}$ reconnaît t.

Réciproquement, considérons un arbre t reconnu par $\mathcal{A}^{\downarrow}$ et montrons qu'il existe un mot $x \in L$ tel que $\text{chaîne}(x) = t$.
Si $t = \varepsilon$ alors $q_0 \in F$ donc $\varepsilon \in L$ et $t = \text{chaîne}(\varepsilon)$.
Si $t \neq \varepsilon$, posons $t = (S, r, \lambda, g, d)$. Si $r \in S_d$, $\varphi(d(r)) = q_1'$ et $\delta(\varphi(d(r)), \lambda(d(r))) = (q_2', q_2')$. Puisque q_2' est un état puits, en considérant un descendant s de $d(r)$ n'ayant pas de fils on obtient une absurdité puisque $q_2' \notin F \cup \{q_1'\}$.
Ainsi, r n'a pas de fils droit, et en procédant récursivement on montre de la même manière que t est un arbre-chaîne. On peut donc poser $S = \{u_1, \ldots, u_l\}$ et $r = u_1$ avec $g(u_{i-1}) = u_i$ et $u_{i-1} \notin S_d$, et définir le mot $x = x_1 \cdots x_l$ avec $x_i = \lambda(u_i)$. Ainsi, $t = \text{chaîne}(x)$, et en posant q_i égale à la composante gauche de $\delta'(u_{i+1}, x_i)$ on obtient un chemin $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \cdots \xrightarrow{x_l} q_l$ menant à un état acceptant dans $\mathcal{A}$, ce qui prouve que $x \in L$.

Question 15. Soit $\mathcal{A}^{\downarrow} = (Q, q_0, F, \delta)$ un automate d'arbres descendant déterministe qui reconnaît chaîne(L). On définit l'automate de mots déterministe complet $\mathcal{A} = (Q, \Sigma, q_0, F, \delta')$ en convenant que pour tout $q \in Q$ et $\alpha \in \Sigma$, si $\delta(q, \alpha) = (q_g, q_d)$ alors $\delta'(q, \alpha) = q_g$.

Si $x = x_1 \cdots x_l \in \Sigma^*$, on définit la suite d'états $q_1, \ldots, q_l$ en posant : pour tout $i \in [\![1, l]\!]$, q_i est la composante gauche de $\delta(q_{i-1}, x_i)$. Par construction, à cette suite d'états est associé un chemin $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \cdots \xrightarrow{x_l} q_l$ dans $\mathcal{A}$.
Or chaîne(x) est reconnu par $\mathcal{A}^{\downarrow}$ si et seulement si $q_l \in F$, et x est reconnu par $\mathcal{A}$ si et seulement si $q_l \in F$. On en déduit que si chaîne(L) est reconnu par $\mathcal{A}^{\downarrow}$ alors L est reconnu par $\mathcal{A}$, et en particulier L est rationnel.

Combiné à la question précédente, nous avons prouvé qu'un langage L est rationnel si et seulement s'il existe un automate d'arbre descendant déterministe qui reconnaît chaîne(L).

Question 16. Il s'agit d'appliquer le lemme de l'étoile (qui n'est plus au programme ; il faut donc le re-démontrer).

Considérons le chemin $q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_0} q_k \xrightarrow{\alpha_1} q_{k+1} \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_1} q_{2k}$. Puisque x est reconnu par $\mathcal{A}$, ce chemin mène à un état $q_{2k} \in \mathrm{F}$. Mais $\mathcal{A}$ n'a que k états, donc il existe $0 < i < j \leqslant k$ tel que $q_i = q_j$ (c'est le principe des tiroirs).

Considérons le mot $y = \alpha_0^j \alpha_0^{j-i} \alpha_0^{k-j} \alpha_1^k = \alpha_0^{k+j-i} \alpha_1^k$. Il est reconnu par $\mathcal{A}$ puisque le chemin $q_0 \xrightarrow{\alpha_0^j} q_j = q_i \xrightarrow{\alpha_0^{j-i}} q_j \xrightarrow{\alpha_0^{k-j}} q_k \xrightarrow{\alpha_1^k} q_{2k}$ est acceptant. Mais ceci est absurde puisque $y \notin \mathrm{L}_{\text{égal}}$.

On en déduit que $\mathrm{L}_{\text{égal}}$ n'est pas rationnel, puis, compte tenu de l'équivalence établie aux deux questions précédentes, qu'il n'existe pas non plus d'automate descendant déterministe reconnaissant chaîne($\mathrm{L}_{\text{égal}}$).

Partie VI. Automates d'arbres ascendants

Question 17. Soit $t = (\mathrm{S}, r, \lambda, g, d)$ un arbre de L_0. Définissons la fonction $\varphi : \mathrm{S} \to \mathrm{Q}$ en posant :

$$\varphi(s) = \begin{cases} q_1 & \text{si } s \text{ ou un de ses descendants est étiqueté par } \alpha_0 \\ q_0 & \text{sinon} \end{cases}$$

Puisque $t \in \mathrm{L}_0$ on a $\varphi(r) = q_1$, et pour tout $u \in \mathrm{S}$,

- si $\varphi(u) = q_0$, alors $\lambda(u) = \alpha_1$ et $\varphi(u) \in \Delta(q_0, q_0, \alpha_1)$, et si $u \in \mathrm{S}_g$ alors $\varphi(g(u)) = q_0$, si $u \in \mathrm{S}_d$ alors $\varphi(d(u)) = q_0$;
- si $\varphi(u) = q_1$ alors :
  - si $u \in \mathrm{S}_g \cap \mathrm{S}_d$ et si $g(u)$ et $d(u)$ possèdent tous deux un descendant étiqueté par α_0 alors $\varphi(u) \in \Delta(q_1, q_1, \lambda(u))$;
  - si $u \in \mathrm{S}_g$ et $u \notin \mathrm{S}_d$ et si $g(u)$ possède un descendant étiqueté par α_0 alors $\varphi(u) \in \Delta(q_1, q_0, \lambda(u))$;
  - si $u \notin \mathrm{S}_g$ et $u \in \mathrm{S}_d$ et si $d(u)$ possède un descendant étiqueté par α_0 alors $\varphi(u) \in \Delta(q_0, q_1, \lambda(u))$;
  - si u ne possède pas de descendant étiqueté par α_0 autre que lui-même, alors $\varphi(u) \in \Delta(q_0, q_0, \alpha_0)$.

t est donc reconnu par $\mathcal{A}_0^{\uparrow}$.

Soit maintenant t un arbre n'appartenant pas à L_0. Si $t = \varepsilon$, t n'est pas reconnu par $\mathcal{A}_0^{\uparrow}$. Et si $t = (\mathrm{S}, r, \lambda, g, d)$ était reconnu, l'application $\varphi : \mathrm{S} \to \mathrm{Q}$ associée à cette reconnaissance vérifierait $\varphi(r) = q_1$, et compte tenu de la table de transition associée à Δ, r posséderait au moins un fils u vérifiant $\varphi(u) = q_1$. Par induction il existerait une feuille v vérifiant $\varphi(v) = q_1$, ce qui est absurde puisque $q_1 \notin \Delta(q_0, q_0, \alpha_1)$.
De ceci il résulte que $\mathcal{L}(\mathcal{A}_0^{\uparrow}) = \mathrm{L}_0$.

Question 18. Posons $\mathcal{A}^{\downarrow} = (\mathrm{Q}, q_0, \mathrm{F}, \delta)$, et définissons l'automate ascendant $\mathcal{A}^{\uparrow} = (\mathrm{Q}, \mathrm{F}, \{q_0\}, \Delta)$ en posant :

$$\forall (q_1, q_2) \in \mathrm{Q}^2,\ \forall \alpha \in \Sigma, \quad \Delta(q_1, q_2, \alpha) = \Big\{ q \in \mathbb{Q} \;\Big|\; \delta(q, \alpha) = (q_1, q_2) \Big\}.$$

Soit $t \in \mathrm{L}$. Si $t = \varepsilon$ alors $q_0 \in \mathrm{F}$ donc $\{q_0\} \cap \mathrm{F} \neq \emptyset$ et t est aussi reconnu par $\mathcal{A}^{\uparrow}$. Si $t = (\mathrm{S}, r, \lambda, g, d)$, notons φ la fonction associée à la reconnaissance par $\mathcal{A}^{\downarrow}$ de l'arbre t. Il est alors aisé de montrer que cette même fonction vérifie les conditions nécessaires pour que $\mathcal{A}^{\uparrow}$ reconnaisse t. Ainsi, $t \subset \mathcal{L}(\mathcal{A}^{\uparrow})$.

Réciproquement, considérons un arbre $t \in \mathcal{L}(\mathcal{A}^{\uparrow})$. Si $t = \varepsilon$ alors $q_0 \in \mathrm{F}$ et t est aussi reconnu par $\mathcal{A}^{\downarrow}$. Si $t = (\mathrm{S}, r, \lambda, g, d)$, soit φ une fonction associée à cette reconnaissance par $\mathcal{A}^{\uparrow}$. Là encore, on vérifie que cette même fonction φ vérifie les conditions pour que $\mathcal{A}^{\downarrow}$ reconnaisse t. Ainsi, $t \in \mathrm{L}$.

De ceci il résulte que $\mathcal{L}(\mathcal{A}^{\uparrow}) = \mathrm{L}$ et donc que L est un langage d'arbres rationnel.

Question 19. Le nombre d'états est la taille du vecteur **`final_asc`** :

```
let nombre_etats_asc aa = vect_length aa.finals_asc ;;
```

Question 20. Le nombre de caractères de l'alphabet est la troisième dimension du tableau **`transitions_asc`** :

```
let nombre_symboles_asc aa = vect_length aa.transitions_asc.(0).(0) ;;
```

Question 21. On procède par induction : on calcule la liste des états possibles qu'on peut associer aux fils gauche et droit de la racine ; il faut alors fusionner, pour chacun des couples (q_g, q_d) obtenus, la liste des états $\Delta(q_g, q_d, \lambda(r))$.

```
let rec applique_asc aa = function
  | Vide    -> aa.initiaux_asc
  | Noeud t -> let lg = applique_asc aa t.gauche and ld = applique_asc aa t.droit in
               let p = produit lg ld in
               let l = map (function (g, d) -> aa.transitions_asc.(g).(d).(t.etiquette)) p in
               fusion l ;;
```

Question 22. Il reste alors à vérifier qu'un des états q obtenus par la fonction précédente vérifie $q \in F$:

```
let evalue_asc aa t = exists (function q -> aa.finals_asc.(q)) (applique_asc aa t) ;;
```

Question 23. Dans cette question il s'agit de montrer qu'on peut « déterminiser » un automate ascendant $\mathcal{A}^\uparrow = (Q, I, F, \Delta)$. Nous allons nous inspirer de l'algorithme de déterminisation classique en posant $\mathcal{A}^\uparrow_d = (Q_d, I_d, F_d, \Delta_d)$ avec :

- $Q_d = \mathcal{P}(Q)$;
- $I_d = \{I\}$;
- $F_d = \left\{A \in \mathcal{P}(Q) \,\middle|\, A \cap F \neq \emptyset\right\}$;
- $\forall (A, B) \in \mathcal{P}(Q)^2, \forall \alpha \in \Sigma, \Delta_d(A, B, \alpha) = \left\{ \bigcup_{(q_1, q_2) \in A \times B} \Delta(q_1, q_2, \alpha) \right\}$.

Cet automate ascendant est bien déterministe, et reconnait le même langage d'arbre que $\mathcal{A}^\uparrow$.

Question 24. Une partie de $[\![0, n-1]\!]$ peut être représentée par le graphe de sa fonction caractéristique, lui-même représenté par l'écriture binaire d'un entier compris entre 0 et $2^n - 1$. Ceci donne les fonctions :

```
let rec identifiant_partie = function
  | []   -> 0
  | t::q -> (1 lsl t) + (identifiant_partie q) ;;


let partie_identifiant n =
  let rec aux acc = function
    | 0                  -> []
    | n when n mod 2 = 0 -> aux (acc + 1) (n / 2)
    | n                  -> acc::(aux (acc + 1) (n / 2))
  in aux 0 n ;;
```

Question 25. Il faut maintenant mettre en œuvre la construction décrite à la question 23. Si n est le nombre d'états de $\mathcal{A}^\uparrow$, les états de l'automate déterminisé sont des parties de $[\![0, n-1]\!]$, qui seront représentées par des entiers compris entre 0 et $2^n - 1$ grâce aux fonctions écrites à la question précédente.
Pour plus de lisibilité, nous allons écrire trois fonctions pour calculer respectivement I_d, F_d et Δ_d. Pour cette dernière fonction, il faudra construire un tableau tri-dimensionnel. De base, Caml ne dispose que des fonctions **`make_vect`** (pour les tableaux uni-dimensionnels) et **`make_matrix`** (pour les tableaux bi-dimensionnels). J'ai donc commencé par définir la fonction :

```
let make_trimatrix p q r x =
  let m = make_matrix p q [||] in
  for i = 0 to p-1 do
    for j = 0 to q-1 do
      m.(i).(j) <- make_vect r x
    done
  done ;
  m ;;
```

Voici maintenant les trois fonctions annoncées :

```
let construire_Id aa = [identifiant_partie aa.initiaux_asc] ;;

let construire_Fd aa =
  let n = nombre_etats_asc aa in
  let f = make_vect n false in
  for i = 0 to n-1 do
    if exists (function k -> aa.finals_asc.(k)) (partie_identifiant i) then f.(i) <- true
  done ;
  f ;;

let construire_Deltad aa =
  let n = nombre_etats_asc aa and p = nombre_symboles_asc aa in
  let d = make_trimatrix n n p [] in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      for k = 0 to p-1 do
        let a = partie_identifiant i and b = partie_identifiant j in
        let p = produit a b in
        let l = map (function (q1, q2) -> aa.transitions_asc.(q1).(q2).(k)) p in
        d.(i).(j).(k) <- [identifiant_partie (fusion l)]
      done
    done
  done ;
  d ;;
```

La fonction demandée s'écrit alors :

```
let determinise_asc aa =
  { initiaux_asc = construire_Id aa ;
    finals_asc = construire_Fd aa ;
    transitions_asc = construire_Deltad aa } ;;
```

Question 26. Notons $\mathcal{A}^{\uparrow} = (Q, I, F, \Delta)$ un automate ascendant *déterministe* qui reconnaît L, et définissons $\mathcal{A}_c^{\uparrow} = (Q_c, I_c, F_c, \Delta_c)$ en posant :

$$Q_c = Q, \quad I_c = I, \quad F_c = Q \setminus F, \quad \Delta_c = \Delta.$$

La déterminisation assure l'unicité de la fonction φ reconnaissant un arbre non vide t, et ainsi, $\mathcal{A}_c^{\uparrow}$ est un automate ascendant qui reconnaît le langage d'arbre complémentaire de L.

Question 27. On en déduit la fonction :

```
let complementaire_asc aa =
  let aac = determinise_asc aa in
  { initiaux_asc = aac.initiaux_asc ;
    finals_asc = map_vect (function b -> not b) aac.finals_asc ;
    transitions_asc = aac.transitions_asc } ;;
```

Question 28. Notons pour $i \in \{1, 2\}$, $\mathcal{A}_i^{\uparrow} = (Q_i, I_i, F_i, \Delta_i)$ un automate ascendant qui reconnaît L_i. Quitte à renommer les états on peut supposer Q_1 et Q_2 disjoints. Définissons alors $\mathcal{A}^{\uparrow} = (Q, I, F, \Delta)$ en posant :

$$Q = Q_1 \cup Q_2, \quad I = I_1 \cup I_2, \quad F = F_1 \cup F_2$$

et pour la fonction de transition :

$$\Delta(q_1, q_2, \alpha) = \begin{cases} \Delta_1(q_1, q_2, \alpha) & \text{si } (q_1, q_2) \in Q_1^2 \\ \Delta_2(q_1, q_2, \alpha) & \text{si } (q_1, q_2) \in Q_2^2 \\ \emptyset & \text{sinon} \end{cases}$$

Alors $\mathcal{A}^{\uparrow}$ reconnaît $L_1 \cup L_2$.

Question 29. On en déduit les fonctions :

```
let construire_Iu aa1 aa2 =
  let n = nombre_etats_asc aa1 in
  union aa1.initiaux_asc (map (prefix + n) aa2.initiaux_asc) ;;

let construire_Fu aa1 aa2 =
  concat_vect aa1.finals_asc aa2.finals_asc ;;

let construire_Deltau aa1 aa2 =
  let n1 = nombre_etats_asc aa1 and p1 = nombre_symboles_asc aa1 in
  let n2 = nombre_etats_asc aa2 and p2 = nombre_symboles_asc aa2 in
  let d = make_trimatrix (n1 + n2) (n1 + n2) (max p1 p2) [] in
  for i = 0 to n1 - 1 do
    for j = 0 to n1 - 1 do
      for k = 0 to p1 - 1 do
        d.(i).(j).(k) <- aa1.transitions_asc.(i).(j).(k)
      done
    done
  done ;
  for i = 0 to n2 - 1 do
    for j = 0 to n2 - 1 do
      for k = 0 to p2 - 1 do
        d.(n1+i).(n1+j).(k) <- aa1.transitions_asc.(i).(j).(k)
      done
    done
  done ;
  d ;;

let union_asc aa1 aa2 =
  { initiaux_asc = construire_Iu aa1 aa2 ;
    finals_asc = construire_Fu aa1 aa2 ;
    transitions_asc = construire_Deltau aa1 aa2 } ;;
```

Question 30. De l'égalité $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ et des questions 26 et 28 il résulte que si L_1 et L_2 sont des langages d'arbres rationnels il en est de même de $L_1 \cap L_2$.

Question 31. On en déduit :

```
let intersection_asc aa1 aa2 =
  complementaire_asc (union_asc (complementaire_asc aa1) (complementaire_asc aa2)) ;;
```

Remarque. Il serait plus naturel de définir l'intersection par $\mathcal{A}_i^{\uparrow} = (Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta)$ avec :

$$\Delta((q_1, q_2), (q_1', q_2'), \alpha) = \Delta_1(q_1, q_1', \alpha) \times \Delta_2(q_2, q_2', \alpha)$$

mais la traduction Caml serait délicate vu la contrainte imposée par l'énoncé quant à la représentation des états par des entiers.

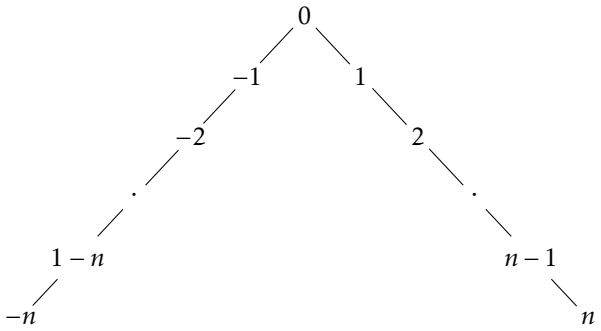
Question 32. Supposons l'existence d'un automate ascendant $\mathcal{A}^{\uparrow} = (Q, I, F, \Delta)$ reconnaissant $L_{\text{impartial}}$, posons $n = |Q|$ et posons $t = (\llbracket -n, n \rrbracket, 0, \lambda, g, d)$ avec :

- $\forall s \in \llbracket -n, n \rrbracket$, $\lambda(s) = \alpha_0$;
- $g : \llbracket 1-n, 0 \rrbracket \to \llbracket -n, n \rrbracket$ est défini par $g(s) = s - 1$;
- $d : \llbracket 0, n-1 \rrbracket \to \llbracket -n, n \rrbracket$ est défini par $d(s) = s + 1$.

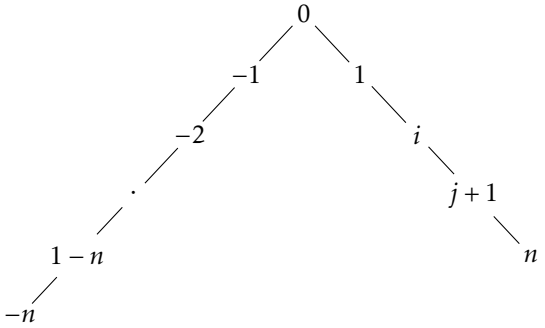
t est impartial donc reconnu par $\mathcal{A}^{\uparrow}$. Soit φ une fonction répondant aux exigences de la page 4.
Puisque $n = |Q|$, le principe des tiroirs affirme l'existence de $0 \leqslant i < j \leqslant n$ tel que $\varphi(i) = \varphi(j)$.
Considérons alors l'arbre $t' = (\llbracket -n, i \rrbracket \cup \llbracket j+1, n \rrbracket, 0, \lambda', g', d')$ défini par :

- $\forall s \in \llbracket -n, i \rrbracket \cup \llbracket j+1, n \rrbracket$, $\lambda(s) = \alpha_0$;
- $g : \llbracket 1-n, 0 \rrbracket \to \llbracket -n, n \rrbracket$ est défini par $g(s) = s - 1$;
- $d : \llbracket 0, i \rrbracket \cup \llbracket j+1, n-1 \rrbracket \to \llbracket -n, n \rrbracket$ est défini par $d(i) = j + 1$ et $d(s) = s + 1$ sinon.

Alors t' n'est pas impartial mais toujours reconnu par $\mathcal{A}^{\uparrow}$, ce qui est absurde.
Le langage d'arbre $L_{\text{impartial}}$ n'est donc pas rationnel.



L'arbre t, étiqueté par ses sommets.



L'arbre t', étiqueté par ses sommets.