Automates d'arbre
un corrigé

Fonctions utilitaires

1. C'est la fonction `mem` de Caml. Dans la version donnée, on tire profit de l'évaluation paressseuse (si $x = y$, le second test n'est pas effectué).

```
let rec contient li x=
   match li with
   [] -> false
  |y::q ->(x=y) or (contient q x) ;;
```

   La complexité est $O(|\texttt{li}|)$ où on note $|\texttt{a}|$ la longueur d'une liste `a`.

2. Pour chaque élément de `l1`, on teste son appartenance à `l2` pour voir s'il faut l'ajouter. Le traitement est bien sûr, en réalité, récursif.

```
let rec union l1 l2=
   match l1 with
   [] -> l2
  |x::q1 -> if (contient l2 x) then union q1 l2
            else x::(union q1 l2);;
```

   La complexité est $O(|\texttt{l1}|.|\texttt{l2}|)$ (on fait $|\texttt{l1}|$ appels à la fonction `contient` avec `l2`).

3. On fait la fusion des listes de la queue et on réunit le résultat avec la tête.

```
let rec fusion l =
   match l with
   [] -> []
  | l1:: q -> union (fusion q) l1 ;;
```

   Avec les notations de l'énoncé, le nombre d'opérations est de l'ordre de

$$\sum_{i=1}^{k} \left(|l_i| \sum_{j=i+1}^{k} |l_j| \right)$$

   que l'on peut majorer par L^2.

4. On écrit une première fonction de type `ajoute : 'a → 'b list → ('a*'b) list` telle que l'appel `ajoute x l` renvoie la liste formée des couples `(x,y)` pour tous les éléments `y` de `l`.

```
let rec ajoute x l=
   match l with
   [] -> []
  |y::q -> (x,y)::(ajoute x q);;
```

   Cette fonction est de complexité $O(|l|)$. La fonction principale s'en déduit grâce à des concaténations.

```
let rec produit l1 l2=
     match l1 with
     [] -> []
    |a1::q1 -> (ajoute a1 l2)@(produit q1 l2);;
```

   La concaténation d'une liste `a` avec une autre a un coût $O(|a|)$ qui ici ne gêne pas puisque c'est aussi le coût d'un appel à `ajoute`. Ici, le coût de la fonction va donc être $O(|\texttt{l1}|.|\texttt{l2}|)$.

Arbres binaires étiquetés

5. La fonction est immédiate.

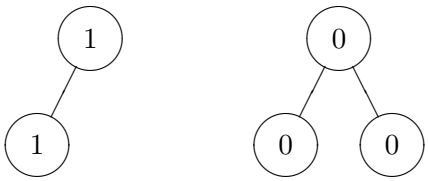
```
let arbre x ag ad = Noeud {etiquette=x;gauche=ag;droit=ad} ;;
```

6. Même chose, il s'agit juste de montrer que l'on sait manipuler un élément de type `Noeud`.

```
let rec taille t =
   match t with
   Vide -> 0
  |Noeud n -> 1 + (taille n.gauche) + (taille n.droit) ;;
```

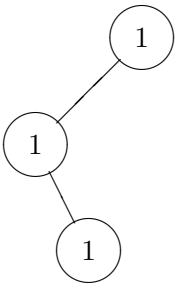
Langages d'arbres

7. Considérons les deux arbres



Celui de gauche est dans L_{chaine} mais dans aucun des trois autres. Celui de droite n'est pas dans L_{chaine} mais est dans chacun des trois autres.

8. L'arbre suivant est impartial (1 fils gauche et 1 fils droit) sans être complet.



Soit $t = (S, r, \lambda, g, d)$ un arbre complet. On a alors $g(S_g) = d(S_d)$ (tout noeud ayant un fils gauche a un fils droit et réciproquement). Mais comme g et d sont injectives, cela induit que $|S_g| = |S_d|$ et donc que l'arbre est impartial.
On peut aussi donner une preuve reposant sur la définition récursive des arbres. Montrons ainsi par récurrence sur la hauteur qu'un arbre complet non vide est impartial (on définit la hauteur d'un arbre comme étant égale à -1 pour l'arbre vide et égale à 1 plus le maximum des hauteurs des fils droit et gauche, en convenant que le fils est vide s'il n'existe pas).

- *<u>Initialisation</u> : le seul arbre de hauteur -1 est l'arbre vide et il est à la fois impartial et complet. On pourrait aussi initialiser avec un arbre de hauteur 1 qui est réduit à sa racine et qui est complet et impartial. Le résultat est donc vrai pour des hauteurs ≤ 0.*
- *<u>Hérédité</u> : supposons le résultat vrai pour tous les rangs $\leq n$ avec $n \geq 0$ fixé. Soit t un arbre complet de hauteur $n+1$. Comme $n+1 \geq 1$, il y a un fils droit ou un fils gauche non vide et comme l'arbre est complet, il y a en fait l'un ET l'autre ; notons les t_g et t_d. Commes ils sont non vides, le nombre de fils droits de t est égal à 1 plus le nombre de fils droits de t_g plus le nombre de fils droits de t_d et on a la même chose pour les fils gauches. Mais t_g et t_d sont des arbres de hauteur $\leq n$ auxquels ont peut appliquer l'hypothèse de récurrence. On en déduit alors le résultat voulu pour t.*

9. Dans un arbre, tout noeud hormis la racine est soit un fils droit soit un fils gauche (de manière exclusive). Le nombre de noeuds d'un arbre est donc égal à 1 plus le nombre de fils droits plus le nombre de fils gauches. Dans le cas d'un arbre impartial non vide t on a alors la taille de t qui est égal à 1 plus deux fois le nombre communs de fils gauche ou droit et qui est donc impaire.
On peut là encore donner une preuve récurrente. Si t est un arbre, je note $|t|$ sa taille, $|t|_g$ le nombre de ses noeuds ayant un fils gauche et $|t|_d$ le nombre de ses fils ayant un fils droit. Je montre alors par récurrence sur la hauteur que pour un arbre t non vide on a $|t| = 1+|t|_g+|t|_d$.
   - *Initialisation : le résultat est immédiatement vrai pour un arbre de hauteur 0 qui est réduit à sa racine.*
   - *Hérédité : supposons le résultat vrai pour tous les arbres non vides de hauteur $\leq n$ pour un $n \geq 0$ donné. Soit t un arbre de hauteur $n+1$. Supposons tout d'abord qu'il a deux fils gauche et droit non vides g et d. On a alors, puisque l'on peut appliquer l'hypothèse de récurrence à g et d,*

$$|t| = 1 + |g| + |d| = 3 + |g|_g + |d|_g + |g|_d + |d|_d$$

Par ailleurs, $|t|_g = 1 + |d|_g + |g|_g$ et $|t|_d = 1 + |d|_d + |g|_d$ et ainsi

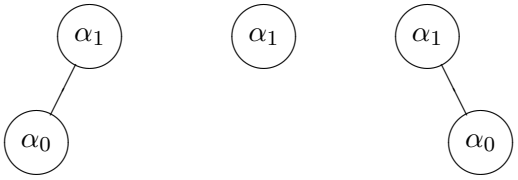
$$|t| = 1 + |t|_g + |t|_g$$

On procède de même quand l'un des deux fils est vide.

Automates d'arbres descendants déterministes

10. Considérons l'automate défini par $Q = \{q_0, q_1, q_R\}$, $F = \{q_0, q_1\}$, $\delta(q_0, \alpha) = (q_0, q_1)$, $\delta(q_1, \alpha) = (q_R, q_R)$ et $\delta(q_R, \alpha) = (q_R, q_R)$. Cet automate descendant déterministe reconnaît le langage L_{chaine}.
Justification (non demandée).
   - *L'arbre vide est bien reconnu car $q_0 \in F$.*
   - *Soit t une chaîne et $u_0, \ldots, u_n$ les noeuds successifs de celle-ci. On construit une application φ convenable. On doit prendre $\varphi(u_0) = q_0$. $g(u_0) = u_1$ existe et $\delta(\varphi(u_0), \alpha) = \delta(q_0, \alpha) = (q_0, q_1)$. On doit choisir $\varphi(u_1) = q_0$. En itérant le raisonnement, on doit choisir $\varphi(u_0) = \cdots = \varphi(u_n) = q_0$. Pour ce choix de φ, les contraintes avec les noeuds possédant des fils gauches sont vérifiées. Aucun $d(u_k)$ n'existe mais $\delta(\varphi(u_k), \alpha) = \delta(q_0, \alpha) = (q_0, q_1)$ et $q_1 \in F$. $g(u_n)$ n'existe pas mais $\delta(\varphi(u_n), \alpha) = \delta(q_0, \alpha) = (q_0, q_1)$ et $q_0 \in F$. t est donc reconnu.*
   - *Soit t un arbre qui n'est pas une chaîne et u_0 sa racine. On descend à gauche dans l'arbre tant qu'il n'y a pas de fils droit. Ceci nous amène au "premier" noeud u_p ayant un fils droit (qui existe puisque t n'est pas une chaîne). Si φ est une application convenable alors, comme ci-dessus, $\varphi(u_0) = \cdots = \varphi(u_p) = q_0$. $u_{p+1} = d(u_p)$ et $\delta(\varphi(u_p), \alpha) = \delta(q_0, \alpha) = (q_0, q_1)$. On a donc $\varphi(u_{p+1}) = q_1$. On a alors $\delta(\varphi(u_{p+1}), \alpha) = (q_R, q_R)$ et on ne "quittera plus q_R" ensuite dans cette partie de l'arbre. Quand on arrivera sur une feuille, on ne parviendra pas à un état terminal. Aucune application φ ne convient et l'arbre n'est pas reconnu.*

11. Supposons, par l'absurde, qu'il existe un automate descendant déterministe $\mathcal{A}^{\downarrow} = (Q, q_0, F, \delta)$ qui reconnaît L_0. Notons $(q_g, q_d) = \delta(q_0, \alpha_1)$. Considérons les arbres



Le premier arbre étant dans L_0, il est reconnu. Sa racine n'admettant pas de fils droit, on a donc $q_d \in F$.
Le second arbre n'étant pas dans L_0, il n'est pas reconnu. Sa racine n'admettant pas de fils, on a q_g ou q_d qui n'est pas dans F et avec ce qui précède, $q_g \notin F$.
Le troisième arbre étant dans L_0, il est reconnu et, comme pour le premier, cela donne $q_g \in F$.
On obtient une contradiction.
ATTENTION : ce raisonnement n'est pas valable si $\Sigma = \{\alpha_0\}$. Dans ce cas, L_0 est constitué de tous les arbres non vides. Dans un tel cas, l'automate descendant déterministe $(\{q_0, q_1\}, q_0, \{q_1\}, \delta)$ avec $\delta(q, \alpha_0) = (q_1, q_1)$ reconnaît bien L_0.

12. On procède récursivement. Le cas de base est immédiat, c'est celui de l'arbre vide. Si l'arbre est un noeud, c'est a priori un peu lourd car le test à faire dépend de la vacuité des fils et il y a ainsi quatre cas à écrire (deux fils vides, un seul des deux, aucun).

```
let rec applique_desc add q t =
   match t with
   Vide -> add.finals_desc.(q)
  |Noeud n -> let (qg,qd)=add.transitions_desc.(q).(n.etiquette) in
              let bg=add.finals_desc.(qg) and bd=add.finals_desc.(qd) in
              (n.droit=Vide && n.gauche=Vide && bg && bd)
             ||(n.droit=Vide && bd && applique_desc add qg n.gauche)
             ||(n.gauche=Vide && bg && applique_desc add qd n.droit)
             ||(applique_desc add qg n.gauche && applique_desc add qd n.droit);;
```

En réalité, le résultat renvoyé dans le cas `Vide` est cohérent avec les appels récursifs effectués et on peut se contenter d'écrire

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

13. On applique la fonction précédente en partant de l'état initial.

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

Automates descendants et langages rationnels de mots

14. Soit $x \in L$ et t la chaîne associé. Si $x = \varepsilon$, on a $q_0 \in F$ (le mot vide est reconnu par $\mathcal{A}$) et donc t est reconnu par $\mathcal{A}^{\downarrow}$. Sinon, on note $u_1, \ldots, u_l$ les noeuds successifs. Comme dans la justification donnée en question **10**, la seule fonction φ qui peut convenir (quand on veut montrer que t est reconnu) est celle définie par

$$\forall i \in [1, l],\ \varphi(u_i) = \delta^*(q_0, x_1 \ldots x_{i-1})$$

Pour ce choix de φ,
- Les contraintes avec les fils gauches qui existent sont vérifiées.
- Aucun $d(u_k)$ n'existe mais $\delta'(\varphi(u_k), x_k) = (\delta^*(q_0, x_1 \ldots x_k), q'_1)$ et q'_1 est terminal.
- $g(u_l)$ n'existe pas mais de même $\delta'(\varphi(u_l), x_l) = (\delta^*(q_0, x_1 \ldots x_l), q'_1)$ et $\delta^*(q_0, x_1 \ldots x_l) \in F$ car x est reconnu par $\mathcal{A}$. C'est donc aussi un état terminal de $\mathcal{A}^{\downarrow}$.

Tous les éléments de $chaine(L)$ sont ainsi reconnus.
Réciproquement, soit t un arbre reconnu. Si $t = \varepsilon$ alors $q_0 \in F \cup \{q'_1\}$ et comme $q_0 \neq q'_1$, $q_0 \in F$ ce qui montre que $\varepsilon \in L$. On a donc bien, dans ce cas, $t \in chaine(L)$. Sinon, t possède une

racine que l'on note u_1 et il existe une application convenable φ. On descend dans l'arbre à gauche tant qu'il n'y a pas de fils droit. Ceci nous amène à un noeud u_p et on a forcément

$$\forall i \in [1,p],\ \lambda(u_i) = x_i \ \text{ et } \ \varphi(u_i) = \delta^*(q_0, x_1 \dots x_{i-1})$$

- Si u_p n'a pas de fils alors $t = chaine(x)$ avec $x = x_1 \dots x_p$. De plus, $\delta'(\varphi(u_i), \lambda(u_i)) = (\delta^*(q_0, x), q'_1)$ et les deux coordonnées sont dans $F \cap \{q'_1\}$ (car t est reconnu) ce qui montre que $\delta^*(q_0, x) \in F$ et donc que $x \in L$ ou encore que $t \in chaine(L)$.
- Sinon u_p a un fils droit que l'on note u_{p+1}. $\delta'(\varphi(u_p), \lambda(u_p)) = (\delta^*(q_0, x_1 \dots x_p), q'_1)$ et donc $\varphi(u_{p+1}) = q'_1$. On a alors $\delta'(\varphi(u_{p+1}), \lambda(u_{p+1})) = (q'_2, q'_2)$ et on ne "quittera plus q'_2" ensuite dans cette partie de l'arbre ce qui contredit le fait que t est reconnu. Ce cas est donc impossible.

15. Supposons que $L \subset \Sigma^*$ est tel que $chaine(L)$ est reconnu par un automate descendant déterministe $\mathcal{A}^\downarrow = (Q, q_0, F, \delta)$. On considère l'automate déterministe $\mathcal{A}' = (Q, \Sigma, q_0, F, \delta')$ où $\delta'(q, \alpha)$ est la première coordonnée de $\delta(q, \alpha)$. Le mot vide est reconnu par $\mathcal{A}'$ si et seulement si $q_0 \in F$ ce qui est la condition nécessaire et suffisante pour que l'arbre vide soit reconnu par $\mathcal{A}^\downarrow$. Soit maintenant $x = x_1 \dots x_p$ un mot non vide et $t = chaine(x)$ dont on note $u_1, \dots, u_p$ les noeuds successifs. On veut montrer que x est reconnu par $\mathcal{A}'$ si et seulement si t est reconnu par $\mathcal{A}^\downarrow$.
    - Supposons que t est reconnu par $\mathcal{A}^\downarrow$ et notons φ la fonction associée. Comme ci-dessus, on montre que
    $$\varphi(u_i) = \delta^*(q_0, x_1 \dots x_{i-1})$$
    et comme u_p n'a pas de fils gauche, on a ensuite (comme ci-dessus) $\delta^*(q_0, x) \in F$ c'est à dire que x est reconnu par $\mathcal{A}'$.
    - Réciproquement, si x est reconnu par $\mathcal{A}'$, on pose $\varphi(u_i) = \delta^*(q_0, x_1 \dots x_{i-1})$ et cette fonction permet de montrer que t est reconnu par $\mathcal{A}^\downarrow$.
16. $x = \alpha_0^k \alpha_1^k$ est reconnu par $\mathcal{A}_{egal}$ et il y a donc un chemin dans cet automate du type
    $$q_0 \xrightarrow{\alpha_0^k} q \xrightarrow{\alpha_1^k} q'' \in F$$
    où q_0 est l'état initial. La première partie du chemin visite $k+1$ états et, d'après le lemme des tiroirs, il y en a donc forcément deux égaux. La première partie du chemin contient donc une "boucle". En la supprimant, on obtient un chemin de l'état initial vers un état final qui comporte strictement plus de α_1 que de α_0 ce qui est contradictoire.
    Aucun automate déterministe ne reconnaît L_{egal}. La contraposée de la question **15** montre alors que $chaine(L_{egal})$ n'est reconnu par aucun automate descendant déterministe.

Automates d'arbres ascendants

17. Soit t un arbre non vide (l'arbre vide n'est pas reconnu et est hors de L_0 ; on exclut ce cas dans la suite de la question). Supposons t reconnu et notons φ une fonction convebale.
    - Si u est une feuille étiquetée par α_0 alors $\varphi(u) \in \Delta(q_0, q_0, \alpha_0) = q_1$. Si u est une feuille étiquetée par α_1 alors $\varphi(u) \in \Delta(q_0, q_0, \alpha_1) = q_0$. La numérotation des feuilles est donc imposée.
    - Si u est un noeud qui n'est pas une feuille et qui possède deux fils g et d aors la valeur de $\varphi(u)$ est imposée par les valeurs de $\varphi(g)$, $\varphi(d)$ et de $\lambda(u)$ car pour tout choix de $q, q' \in Q$ et $x \in \Sigma$, $\Delta(q, q', x)$ est un singleton. Il en va de même quand u ne possède qu'un seul fils.

    φ est donc parfaitement définie. Réciproquement, pour ce choix de φ, la condition (ii) est respectée et l'arbre sera reconnu si et seulement si $\varphi(r) = q_1$.
    On montre maintenant en deux temps que t est reconnu si et seulement si il est dans L_0.

- Supposons que t ne contienne que des étiquettes égales à α_1. Supposons, par l'absurde, que t soit reconnu. On a $\varphi(r) = q_1$ et $q_1 \notin \Delta(q_0, q_0, \alpha_1) = \{q_0\}$. Ceci nous montre que r a au moins un fils et que l'un de ces deux fils a une image par φ égale à q_1. On en déduit en itérant le processus que l'une des feuilles f aura une image par φ égale à q_1. Et comme $q_1 \notin \Delta(q_0, q_0, \alpha_1)$, ceci contredit le fait que t est reconnu. Les seuls arbres qui peuvent être reconnus sont ceux de L_0.
- Supposons que $t \in L_0$ et notons u un noeud d'étiquette α_0. Comme $\Delta(q, q', \alpha_0)$ ne contient jamais q_1 on a $\varphi(u) = q_1$. Mais comme $\Delta(q_1, q, x)$ ou $\Delta(q, q_1, x)$ est toujours égal à $\{q_1\}$, le père v de u vérifie aussi $\varphi(v) = q_1$. En itérant le processus, on aura $\varphi(r) = q_1$ et t est donc reconnu.

18. Soit L un langage d'arbre. Il existe donc un automate descendant déterministe $\mathcal{A}^{\downarrow} = (Q, q_0, F, \delta)$ qui reconnaît L. Posons

$$\mathcal{A}^{\uparrow} = (Q, F, \{q_0\}\Delta) \text{ avec } \Delta(q_g, q_d, \alpha) = \{q \in Q / \ \delta(q, x) = (q_g, q_d)\}$$

Montrons que $\mathcal{L}(\mathcal{A}^{\uparrow}) = L$, ce qui montrera que L est un langage d'arbres rationnel.
- Soit $t \in L$; t est reconnu par $\mathcal{A}^{\downarrow}$. Si $t = \varepsilon$ alors $q_0 \in F$ et on a donc $F \cap \{x_0\} \neq \emptyset$ ce qui indique que t est reconnu par $\mathcal{A}^{\uparrow}$. Si t n'est pas vide, on lui associe une fonction φ qui vérifie les propriété (i) et (ii) des automates descendant. Montrons que cette même fonction φ vérifie les propriétés (i) et (ii) des arbres ascendants.
  (i) $\varphi(r) = q_0 \in \{q_0\}$.
  (ii) Soit $u \in S$. Posons $(q_g, q_d) = \delta(\varphi(u), \lambda(u))$; on a donc $\varphi(u) \in \Delta(q_g, q_d, \lambda(u))$. Si $g(u)$ est défini, $\varphi(g(u)) = q_g$ et sinon $q_g \in F$. Si $d(u)$ est défini, $\varphi(d(u)) = q_d$ et sinon $q_d \in F$.
- Si $t \in \mathcal{L}(\mathcal{A}^{\uparrow})$, on distingue de même le cas $t = \varepsilon$ et le cas t non vide. La preuve est essentiellement la même et est omise.

19. Le nombre d'états est donné par la taille du tableau donnant les éléments de F.

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

20. Le nombre de lettres est la "troisième dimension" du tableau des transitions.

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

21. Notons α l'étiquette de la racine. Il est possible de choisir $\phi(r) = q$ s'il existe des états convenables q_g et q_d pour les fils gauche et droit (en supposant, pour l'instant, qu'il existent) tels que $q \in \Delta(q_g, q_d, \alpha)$. En notant P_t l'ensemble cherché pour un arbre t, on a donc

$$P_t = \bigcup_{(q_g, q_d) \in P_{g(t)} \times P_{d(t)}} \Delta(q_g, q_d, \lambda(t))$$

Si, par exemple, il n'y a pas de fils gauche, il faut remplacer (définition de φ) $P_{g(t)}$ par l'ensemble I. Ceci explique le choix du cas de base donné par l'énoncé. On procède donc de la manière suivante :
- On fait un appel récursif sur la gauche et la droite pour obtenir les listes `lg` et `ld` des états permis à gauche et droite et on en fait le produit cartésien qui donne une liste `prod`.
- Pour chaque élément (i, j) de `prod`, on a une liste $\Delta(i, j, \alpha)$ (α est l'étiquette de notre arbre) et on doit réunir sans doublon ces listes. J'utilise pour cela une fonction $(i, j) \mapsto \Delta(i, j, \alpha)$ que j'applique à chaque élément de `prod` grâce à la fonctionnelle `map`. Il reste alors à fusionner ces listes.

```
let rec applique_asc aa t =
    match t with
    Vide -> aa.initiaux_asc
   |Noeud n ->
```

```
    let lg=applique_asc aa n.gauche in
    let ld=applique_asc aa n.droit in
    let prod=produit lg ld in
    let l=map (fun (i,j) -> aa.transitions_asc.(i).(j).(n.etiquette)) prod
    in fusion l ;;
```

22. La question précédente nous donne la liste des états convenables associables à la racine. Il suffit de voir si parmi ceux-ci, il en est un qui est final. La fonction auxiliaire `contient_final : int list → bool` indique si l'un des éléments de la liste est final.

```
let evalue_asc aa t =
   let rec contient_final l =
      match l with
      [] -> false
     |i::q -> aa.finals_asc.(i) || contient_final q
   in contient_final (applique_asc aa t ) ;;
```

23. S'il existe un automate ascendant déterministe qui reconnaît L alors L est, par définition, un langage d'arbres rationnel (le cas déterministe n'étant qu'un cas particulier du cas général). Réciproquement, supposons que L est un langage d'arbres reconnu par l'automate ascendant $\mathcal{A}^{\uparrow} = (Q, I, F, \Delta)$. On cherche un un automate ascendant déterministe qui reconnaît L, c'est à dire à "déterminiser" $\mathcal{A}^{\uparrow}$. Prenons modèle sur l'algorithme de déterminisation du cours et posons $\mathcal{D}^{\uparrow} = (Q', I', F', \Delta')$ avec

$$Q' = \mathcal{P}(Q),\ I' = \{I\},\ F' = \{A \in Q' = \mathcal{P}(Q) /\ F' \cap F \neq \emptyset\}$$

$$\forall A, B \in \mathcal{P}(Q),\ \forall \alpha \in \Sigma,\ \Delta'(A, B, \alpha) = \left\{ \bigcup_{(q,q') \in A \times B} \Delta(q, q', \alpha) \right\}$$

On a alors bien un automate ascendant déterministe. Montrons qu'il reconnaît exactement le langage L.

- Soit t un élément de L c'est à dire un arbre reconnu par $\mathcal{A}^{\uparrow}$.
  Si $t = \varepsilon$ alors $I \cap F \neq \emptyset$ et donc $I \in F'$ ce qui indique que $I' \cap F' \neq \emptyset$ et que $t = \varepsilon$ est reconnu par $\mathcal{D}$.
  Si $t \neq \varepsilon$, on peut trouver une bonne application $\varphi : S \to Q$ associée. On considère l'application $\psi : S \to Q' = \mathcal{P}(Q)$ telle que $\psi(s)$ est égal au résultat de `applique_asc` avec le sous-arbre de racine s. On doit vérifier deux choses pour ψ.
  (i) On a $\varphi(r) \in \psi(r)$ et donc $\varphi(r) \in \psi(r) \cup F \neq \emptyset$ c'est à dire $\psi(r) \in F'$.
  (ii) Soit $u \in S$. D'après la construction de `applique_asc`, on a $\psi(u) = \Delta'(G, D, \lambda(u))$ avec G et D égaux à $\psi(g(u))$ et $\psi(d(u))$ s'ils existent et I sinon. Ceci correspond exactement à la propriété voulue.
- Soit t un arbre reconnu par $\mathcal{D}$.
  Si $t = \varepsilon$ alors $I' \cap F' \neq \emptyset$ c'est à dire $I \in F'$ ou encore $I \cap F \neq \emptyset$ et donc $t = \varepsilon$ est reconnu par $\mathcal{A} \uparrow$.
  Si $t \neq \varepsilon$, on peut trouver une bonne application $\psi : S \to \mathcal{P}(Q)$. On a $\psi(r) \in F' \neq \emptyset$ c'est à dire $\psi(r) \cap F \neq \emptyset$. On choisit (arbitrairement) $\varphi(r)$ dans $\psi(r) \cap F$ (on vient de voir que c'est possible). De plus, il existe $G, D \in \mathcal{P}(Q)$ tels que $\psi(r)$ est l'unique élément de $\Delta'(G, D, \lambda(u))$ c'est à dire $\psi(r) = \bigcup_{(q,q') \in G \times D} \Delta(q, q', \alpha)$. Comme $\varphi(r)$ est dans cet ensemble, il existe q_g et q_d dans G et D tels que $\varphi(r) \in \Delta(q_g, q_d, \alpha)$.
  Si $g(r)$ existe, on choisit $\varphi(g(r)) = q_g$. Sinon, $G \in I'$ c'est à dire $G = I$ et donc $q_g \in I$.
  Si $d(r)$ existe, on choisit $\varphi(d(r)) = q_d$. Sinon, $D \in I'$ c'est à dire $D = I$ et donc $q_d \in I$.
  On peut alors poursuivre la construction de φ pour la définir sur S tout entier et obtenir une application convenable par conctruction qui montre que t est reconnu par $\mathcal{A}^{\uparrow}$.

24. Un entier de $[|0, 2^n-1|]$ est codable de manière unique sur n bits ce qui nous donne une bijection entre $[|0, 2^{n-1}|]$ et les entiers binaires qui s'écrivent $b_{n-1}b_{n-2}\dots b_1b_0$. Par ailleurs l'application qui envoie l'entier binaire $b_{n-1}b_{n-2}\dots b_1b_0$ sur $\{k/\ b_k = 1\}$ est une bijection de ces entiers binaires dans les parties de $[|0, n-1|]$. La composée de ces applications donne une bijection entre $[|0, 2^n-1|]$ et les parties de $[|0, n-1|]$.
Par exemple, pour $n = 8$,à la partie $\{3, 6\}$ on associe l'entier $2^3 + 2^6 = 144$. Réciproquement, on peut écrire 189 sur 8 bits sous la forme 10111101 (car $189 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$) et on lui associe la partie $\{0, 2, 3, 4, 5, 7\}$.
Pour la fonction `identifiant_partie`, chaque élément de la liste argument est un entier k et on doit ajouter toutes toutes les quantités 2^k.

```
let rec identifiant_partie l =
   match l with
   [] -> 0
  |k::q -> (1 lsl k) + (identifiant_partie q);;
```

Pour la fonction `partie_identifiant` on doit plus ou moins décomposer l'argument en base 2 et renvoyer la liste des positions où l'on trouve un bit égal à 1. J'utilise pour cela une fonction auxiliaire `construire : int → int → int list`. Dans l'appel `construire n k`, l'argument `k` nous indique "le numéro du prochain bit que l'on regarde" (le résultat renvoyé est donc la listes des $i + k$ où les b_i sont les bits non nuls de n).

```
let partie_identifiant n =
   let rec construire n k =
   match n with
   0 -> []
  |_ -> if n mod 2 = 0 then construire (n/2) (k+1)
        else k::(construire (n/2) (k+1))
  in construire n 0 ;;
```

25. On connaît $I \subset Q$, $F \subset Q$ et Δ. Il s'agit de construire I', F' et Δ' comme expliqué en question **23**.
    - On note `n`$= |Q|$, `nl`$= |\Sigma|$ et on calcule `N`$= |\mathcal{P}(Q)| = 2^n$ (étape 1)
    - I' est constitié d'un unique élément qui correspond à la partie I et sera donc constitué d'une liste à 1 élément. On calcule cet élément `init` à l'étape 2.
    - F' est représenté par un tableau à `N` éléments booléens. A l'étape 3, on l'initialise puis on le remplit. La case numéro i correspond à une partie A connue sous forme de liste et on cherche si cette liste contient unn élément final de `aa`. On reprend pour cela la fonction auxiliaire `contient_final` de la question **22**.
    - Δ' est représenté comme un tableau à trois dimensions, c'est à dire une matrice dont chaque élément est un tableau. On commence par initialiser un tel tableau (**attention** : à ce niveau, toutes les cas `D`.$(i).(j)$ ont des contenus physiquement égaux ; si on écrit `D`.$(0).(0).(1) \leftarrow [0]$ on modifie en fait TOUTES les cases `D`.$(i).(j).(1)$; à la ligne commentée `!!!` on place un nouveau tableau dans la case numéro i, j). Ceci est l'étape 4.
    - A l'étape 5, on remplit chaque case de `D`, d'où une triple boucle. `D`.$(i).(j).(k)$ doit recevoir une liste contenant un unique élément qui est l'identifiant de

$$\bigcup_{(a,b)\in A\times B} \Delta(a, b, \alpha)$$

avec A qui est la partie associée à l'entier i et B celle associée à l'entier j. On calcule donc A puis B puis $A \times B$ et pour chaque valeur de k, on construit la réunion ci-dessus (comme en question **21**). C'est la liste contenant le numéro de la partie construite que l'on place dans la case.

```
let determinise_asc aa =
(* \'etape 1 *)
   let n=nombre_etats_asc aa in
   let nl=nombre_symboles_asc aa in
   let N=(1 lsl n) in
(* \'etape 2 *)
   let init=identifiant_partie aa.initiaux_asc in
(* \'etape 3 *)
   let fin=make_vect N false in
   let rec contient_final l =
      match l with
      [] -> false
     |i::q -> aa.finals_asc.(i) || contient_final q in
   for i=0 to N-1 do
      fin.(i) <- contient_final (partie_identifiant i)
      done ;
(* \'etape 4 *)
   let D=make_matrix N N (make_vect nl []) in
(* \'etape 5 *)
   for i =0 to N-1 do
     for j=0 to N-1 do
       D.(i).(j) <- make_vect nl [] ;   (* !!! *)
       let A=partie_identifiant i in
       let B=partie_identifiant j in
       let prod=produit A B in
       for k=0 to nl-1 do
          let l=map (fun (a,b) -> aa.transitions_asc.(a).(b).(k)) prod in
          D.(i).(j).(k) <- [identifiant_partie (fusion l)]
          done;
       done;
       done;
(* On renvoie le bon automate ascendant d\'eterministe *)
   {initiaux_asc = [init] ;
    finals_asc = fin ;
    transitions_asc = D};;
```

26. Soit L un langage d'arbres rationnel. Quitte à déterminiser, il est reconnu par un automate déterministe ascendant $\mathcal{A}^\uparrow = (Q, \{q_0\}, F, \Delta)$. Posons $\mathcal{B}^\uparrow = (Q, \{q_0\}, Q \setminus F, \Delta)$. Si t est un arbre non vide, il existe une unique application φ qui vérifie le point (ii) de la définition d'un arbre reconnu par $\mathcal{A}$. Si t est non reconnu, c'est à dire hors de L alors $q_0 \notin F$; la même application φ montre qu'alors t est reconnu par $\mathcal{B}^\uparrow$. Par symétrie des rôles, on a la réciproque. Enfin, on montre immédiatement que les conditions de reconnaissance de ε par $\mathcal{A}^\uparrow$ et $\mathcal{B}^\uparrow$ sont des négations l'une de l'autre. Finalement, $\mathcal{B}^\uparrow$ reconnaît $\mathcal{T}^\Sigma \setminus L$.

27. Il suffit de déterminiser et de changer les valeurs booléennes du tableau des éléments finals de l'automate obtenu.

```
let complementaire_asc aa =
   let bb=determinise_asc aa in
   for i=0 to (nombre_etats_asc bb) do
      bb.finals_asc.(i) <- not (bb.finals_asc.(i))
      done;
   bb;;
```

28. Notons $\mathcal{A}_i^\uparrow = (Q_i, I_i, F_i, \Delta_i)$ un automate ascendant reconnaissant L_i. Quitte à renommer les états, on peut supposer que $Q_1 \cap Q_2 = \emptyset$. Notons alors $\mathcal{A}^\uparrow = (Q_1 \cup Q_2, I_1 \cup I_2, F_1 \cup F_2, \Delta)$ où la restriction de Δ à $Q_i \times Q_i \times \Sigma$ est égale à Δ_i et où $\Delta(q_1, q_2, x) = \emptyset$ quand $q_1 \in Q_1$ et $q_2 \in Q_2$.
Si t est un arbre non vide reconnu par $\mathcal{A}_i^\uparrow$ il lui est associé une application φ_i et la même application montre que t est reconnu par $\mathcal{A}^\uparrow$.
Si t est un arbre non vide reconnu par $\mathcal{A}^\uparrow$, il lui est associée une application φ. $\varphi(r) \in F = F_1 \cup F_2$. Si $\varphi(r) \in F_i$ alors on montre que φ est à valeurs dans Q_i et cette application permet de justifier que t est reconnu par $\mathcal{A}_i^\uparrow$.
On montre enfin que ε est reconnu par $\mathcal{A}^\uparrow$ si et seulement il est reconnu par l'un des $\mathcal{A}_i^\uparrow$.
Toute ceci indique que $L_1 \cap L_2 = \mathcal{L}(\mathcal{A}^\uparrow)$ et que ce langage est un langage rationnel d'arbres.

29. On met en oeuvre la méthode décrite dans la question précédente. Comme dans la fonction de déterminisation, il faut être attentif aux problèmes d'identité physiques des tableaux.

```
let union_asc aa1 aa2 =
   let n1=nombre_etats_asc aa1 in
   let n2=nombre_etats_asc aa2 in
   let nl=nombre_symboles_asc aa1 in
(* cr\'eation du tableau r\'eunion des \'etats finaux *)
   let F=make_vect (n1+n2) false in
   for i=0 to n1 do F.(i) <- aa1.finals_asc.(i) done;
   for i=0 to n2 do F.(n1+i) <- aa2.finals_asc.(i) done;
(* cr\'eation de la matrice de transition : un coquille vide *)
   let U=make_matrix (n1+n2) (n1+n2) (make_vect nl []) in
(* On met un tableau vide diff\'erent dans chaque case de la matrice *)
   for i=0 to n1+n2-1 do
     for j=0 to n1+n2-1 do
       U.(i).(j) <- make_vect nl [] ;
       done;
     done ;
(* On remplit le quart sup\'erieur gauche *)
   for i=0 to n1-1 do
     for j=0 to n1-1 do
       for k=0 to nl-1 do
          U.(i).(j).(k) <- aa1.transitions_asc.(i).(j).(k) ;
          done;
       done ;
     done;
(* On remplit le quart sup\'erieur droit *)
   for i=0 to n2-1 do
     for j=0 to n2-1 do
       for k=0 to nl-1 do
          U.(n1+i).(n1+j).(k) <- aa2.transitions_asc.(i).(j).(k) ;
          done;
       done ;
     done;
(* On renvoie l'automate *)
   {initiaux_asc =aa1.initiaux_asc @ aa2.initiaux_asc ;
    finals_asc = F;
    transitions_asc = U};;
```

30. Notons $\mathcal{A}_i^\uparrow = (Q_i, I_i, F_i, \Delta_i)$ un automate ascendant reconnaissant L_i. On considère l'automate

ascendant

$$\mathcal{A}^{\uparrow} = (Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta)$$

où Δ est définie par

$$\forall (q_1, q_2), (q_1', q_2') \in Q_1 \times Q_2,\ \forall x \in \Sigma,\ \Delta((q_1, q_2), (q_1', q_2'), x) = \Delta_1(q_1, q_1', x) \times \Delta_2(q_2, q_2', x)$$

ε est reconnu par $\mathcal{A}^{\uparrow}$ ssi $F_1 \times F_2 \cap I_1 \times I_2 \neq \emptyset$ c'est à dire ssi $F_1 \cap I_1$ et $F_2 \cap I_2$ sont non vide c'est à dire ssi $\varepsilon \in L_1 \cap L_2$.
Soit t un arbre non vide reconnu par $\mathcal{A}^{\uparrow}$. Il existe une application $\varphi : S \to Q_1 \times Q_2$ associée. Les fonctions coordonnées φ_1 et φ_2 vont permettre de montrer que t est reconnu par $\mathcal{A}_1 \uparrow$ et $\mathcal{A}_2^{\uparrow}$.
Réciproquement, si t est reconnu par $\mathcal{A}_1^{\uparrow}$ et $\mathcal{A}_2^{\uparrow}$, on a deux fonctions φ_1 et φ_2 et $\varphi : s \mapsto (\varphi_1(s), \varphi_2(s))$ montre que t est reconnu par $\mathcal{A}^{\uparrow}$.
Finalement, $L_1 \cap L_2$ est un langage d'arbres rationnel puisque c'est l'ensemble des arbres reconnus par $\mathcal{A}^{\uparrow}$.
Une façon alternative de procéder est de remarquer que (en notant $\overline{L} = \mathcal{T}^{\Sigma} \setminus L$)

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

et d'utiliser les questions **26** et **28**.

31. Passer par l'automate produit est certainement possible mais délicat à implémenter. Je vais donc utiliser la seconde technique (algorithmiquement moins bonne à mon avis car elle impose plusieurs déterminisations cachées dans les passages au complémentaire).

```
let intersection_asc aa1 aa2 =
    let bb1 = complementaire_asc aa1 in
    let bb2 = complementaire_asc aa2 in
    complementaire_asc (union_asc bb1 bb2);;
```

32. Supposons, par l'absurde, que $L_{impartial}$ soit un langage rationnel d'arbres. Il existe alors un automate ascendant déterministe $\mathcal{A}^{\uparrow} = (Q, \{q_1\}, F; \Delta)$ le reconnaissant. Notons k le cardinal de Q et considérons l'arbre constitué de deux "chaînes" à droite et à gauche chacune constituée de $k+1$ noeuds et tel que tous les noeuds soient étiquetés par la même lettre x. La racine a un fils gauche qui est une chaîne à gauche et un fils droit qui est une chaîne à droite. Cet arbre est bien sûr impartial et il est donc reconnu ; on note φ une application associée. Notons $q_{k+1}, \ldots, q_1$ les images par φ des fils gauches successifs ; comme l'automate est déterministe, on peut les identifier :

$$\{q_1\} = \Delta(q_0, q_0, x) \quad \text{et} \quad \forall i \in [|2, k+1|],\ \{q_i\} = \Delta(q_{i-1}, q_0, x)$$

$q_1, \ldots, q_{k+1}$ sont des éléments d'un ensemble de cardinal k et ne peuvent être deux à deux distincts. Il existe donc $i < j$ tels que $q_i = q_j$. L'arbre t' obtenu en remplaçant le fils gauche du noeud associé à q_j par celui associé à q_i (éventuellement l'arbre vide) est alors lui aussi reconnu (même application φ) mais n'est pas impartial. Ceci est bien sûr une contradiction.
Remarque : on n'a pas vraiment besoin d'un automate déterministe pour faire cette preuve mais cela me semble plus simple à visualiser.