1 Premiers exemples

Question 1
$\mathcal{A}_1$ reconnaît les mots de longueur impaire.

Question 2
$\mathcal{A}_2$ reconnaît les mots qui contiennent un nombre impair de b.

Question 3
Le langage reconnu par $\mathcal{A}_1$ peut être dénoté par `(a|b).((a|b).(a|b))*`.
On a éliminé quelques parenthèses.

Question 4
Le langage reconnu par $\mathcal{A}_2$ peut être dénoté par `a*.b.(a*.b.a*.b)*.a*`.

Question 5

```
let Q = (2, [|(0, 1); (1, 0)|], [|false; true|]);;
```

2 États accessibles d'un automate

Question 6

```
let numero k liste =
  let reponse  = Array.make k (-1) in
  let rec aux reste pos =
    match reste  with
    |[] -> reponse
    |t::q -> reponse.(t) <- pos;
             aux q (pos +1) in
  aux liste 0;;
```

La fonction indique la dernière position de l'élément dans la liste.

Question 7
Comme la question l'indique on parcourt le graphe en profondeur, la formulation récursive est la plus simple. On accumule les sommets parcourus dans une liste (référencée), l'ordre sera inversé donc on retourne la liste à la fin.

```
let etats_accessibles q =
  let n, delta, f = q in
  let vus = Array.make n false in
  let sommets = ref [] in
  let rec aux s =
    if not vus.(s)
    then begin
           vus.(s) <- true;
           sommets := s :: !sommets;
           let succ_a, succ_b = delta.(s) in
           aux succ_a;
           aux succ_b end in
  aux 0 [];
  List.rev !sommets;;
```

Question 8
On part de la liste des sommets accessibles et on construit un nouvel automate dont les états sont ceux de la liste, ils sont donc numérotés par leur position dans la liste. Comme 0 est le premier sommet visité dans le parcours il reste à l'indice 0 dans la liste. Le numéro d'un sommet sera donné par sa valeur dans le tableau calculé par `numero`. Il ne faut pas oublier de convertir aussi les destinations des transitions.

```
let partie_accessible q =
  let n, delta, f = q in
  let acc = etats_accessibles q in
  let num = numero n acc in
  let n1 = List.length acc in
  let delta1 = Array.make n1 (0, 0) in
  let term1 = Array.make n1 false in
  let rec aux reste =
    match reste with
    |[] -> (n1, delta1, term1)
    |t::q -> let s = num.(t) in
             term1.(s) <- term.(t);
             let succ_a, succ_b = delta.(t) in
             delta1.(s) <- (num.(succ_a), num.(succ_b));
             aux q in
  aux acc;;
```

3 Morphismes d'automates

3.1 Exemples de morphismes d'automates

Question 9

— L'état initial E doit être envoyé en l'état initial C.
— Le seul état final, G, doit être envoyé dans un état final, ce ne peut être que D.
— On a $C = \delta_2(C,a) = \delta_2\big(\varphi(E,a)\big) = \varphi\big(\delta_3(E,a)\big) = \varphi(D)$.

On trouve vérifie qu'on a ainsi défini un morphisme d'automates.

s	E	F	G
$\varphi(s)$	C	C	D

Question 10
Les états finaux doivent être envoyés dans l'état final, de même les états non finaux n'ont que le seul état non final comme image possible. Le seul morphisme possible est donc

s	H	I	J	K
$\varphi(s)$	C	C	D	D

Question 11
L'état final doit être envoyé dans l'état final, de même l'état non final doit être envoyé dans l'état non final donc le seul morphisme possible de $\mathcal{A}_1$ vers $\mathcal{A}_2$ est défini par $\varphi(A) = C$ et $\varphi(B) = D$. Mais alors $\delta_2\big(\varphi(A),a\big) = \delta_2(C,a) = C$ est différent de $\varphi\big(\delta_1(A,a)\big) = \varphi(B) = D$: φ ne peut être un morphisme. Il n'y a pas de morphisme de $\mathcal{A}_1$ vers $\mathcal{A}_2$.

Question 12
Par le même argument sur les états finaux et non finaux, le seul morphisme possible de $\mathcal{A}_2$ vers $\mathcal{A}_2$ est défini par $\varphi(L) = \varphi(N) = C$ et $\varphi(M) = D$. Or on a $\delta_2\big(\varphi(N),a\big) = \delta_2(C,a) = C$ différent de $\varphi\big(\delta_5(N,a)\big) = \varphi(M) = D$, il n'existe pas morphisme de $\mathcal{A}_5$ vers $\mathcal{A}_2$.

3.2 Propriétés des morphismes d'automates

Question 13

On considère un morphisme φ de l'automate $\mathcal{A}$ vers l'automate $\mathcal{B}$.
Soit $\mathcal{P}(n)$ la propriété, pour tout état s et pour tout mots u de longueur n,
$\varphi\big(\delta^*_{\mathcal{A}}(s,u)\big) = \delta^*_{\mathcal{B}}\big(\varphi(s),u\big)$.

- $\mathcal{P}(0)$ est valide car ε est le seul mot de longueur 0 et $\varphi\big(\delta^*_{\mathcal{A}}(s,\varepsilon)\big) = \varphi(s) = \delta^*_{\mathcal{B}}\big(\varphi(s),\varepsilon\big)$.
- On suppose $\mathcal{P}(n)$ vérifiée. Soit $u = xw$ un mot de longueur $n+1$ avec w de longueur n.

$$\begin{aligned}\varphi\big(\delta^*_{\mathcal{A}}(s,u)\big) = \varphi\big(\delta^*_{\mathcal{A}}(s,xw)\big) &= \varphi\Big(\delta^*_{\mathcal{A}}\big(\delta_{\mathcal{A}}(s,x),w\big)\Big) \text{ par définition de } \delta^*_{\mathcal{A}}\\ &= \delta^*_{\mathcal{B}}\Big(\varphi\big(\delta_{\mathcal{A}}(s,x)\big),w\Big) \text{ par hypothèse de récurrence}\\ &= \delta^*_{\mathcal{B}}\Big(\delta_{\mathcal{B}}\big(\varphi(s),x\big)\big),w\Big) \text{ d'après la propriété (3) d'un morphisme}\\ &= \delta^*_{\mathcal{B}}\big(\varphi(s),xw)\big)\Big) \text{ par définition de } \delta^*_{\mathcal{B}}\end{aligned}$$

Ainsi $\mathcal{P}(n+1)$ est vérifiée.

On a donc prouvé la propriété pour tout n par récurrence.

En particulier $\varphi\big(\delta^*_{\mathcal{A}}(i_{\mathcal{A}},u)\big) = \delta^*_{\mathcal{B}}\big(\varphi(i_{\mathcal{A}},u)\big) = \delta^*_{\mathcal{B}}(i_{\mathcal{B}},u)$ pour tout mot u d'après la propriété (1).
Or, d'après la propriété (4), $\delta^*_{\mathcal{B}}\big(\varphi(i_{\mathcal{A}},u)\big) = \varphi\big(\delta^*_{\mathcal{A}}(i_{\mathcal{A}},u)\big) \in F_{\mathcal{B}}$ est équivalent à $\delta^*_{\mathcal{A}}(i_{\mathcal{A}},u) \in F_{\mathcal{A}}$ donc $\mathcal{A}$ et $\mathcal{B}$ reconnaissent les mêmes mots, ils définissent le même langage.

Question 14

Si $\mathcal{A}$ et $\mathcal{B}$ ont le même nombre d'élément, toute surjection de l'un vers l'autre est une bijection ; ainsi tout morphisme est bijectif dans ce cas.
On a, en utilisant les propriétés de φ,

1. φ^{-1} est une bijection donc est surjective.
2. $\varphi^{-1}(i_{\mathcal{B}}) = \varphi^{-1}\big(\varphi((i_{\mathcal{A}})\big) = i_{\mathcal{A}}$.
3. $\varphi^{-1}\big(\delta_{\mathcal{B}}(s,x)\big) = \varphi^{-1}\big(\delta_{\mathcal{B}}(\varphi\circ\varphi^{-1}(s),x)\big) = \varphi^{-1}\circ\varphi\big(\delta_{\mathcal{A}}(\varphi^{-1}(s),x)\big) = \delta_{\mathcal{B}}(\varphi^{-1}(s),x)$.
4. $s \in F_{\mathcal{B}} \iff \varphi\circ\varphi^{-1}(s) \in F_{\mathcal{B}} \iff \varphi^{-1}(s) \in F_{\mathcal{A}}$.

Ainsi φ^{-1} est un morphisme d'automates de $\mathcal{B}$ vers $\mathcal{A}$.

Question 15

φ est un morphisme d'automates de $\mathcal{A}$ vers $\mathcal{B}$ et ψ est un morphisme d'automates de $\mathcal{B}$ vers $\mathcal{C}$.

1. $\psi\circ\varphi$ est une composée de fonctions surjectives donc est surjective.
2. $\psi\circ\varphi(i_{\mathcal{A}}) = \psi(i_{\mathcal{B}}) = (i_{\mathcal{C}})$.
3. $\psi\circ\varphi\big(\delta_{\mathcal{A}}(s,x)\big) = \psi\big(\delta_{\mathcal{B}}(\varphi(s),x)\big) = \delta_{\mathcal{C}}\big(\psi\circ\varphi(s),x)\big)$.
4. $s \in F_{\mathcal{A}} \iff \varphi(s) \in F_{\mathcal{B}} \iff \psi\circ\varphi(s) \in F_{\mathcal{C}}$.

Ainsi $\psi\circ\varphi$ est un morphisme d'automates de $\mathcal{A}$ vers $\mathcal{C}$.

3.3 Existence de morphismes d'automates entre automates accessibles

Question 16

S'il existe un morphisme d'automates, φ (ne vérifiant pas nécessairement (1)), de $\mathcal{A}$ vers $\mathcal{B}$ avec $\mathcal{B}$ accessible alors, pour tout état s de $\mathcal{B}$, il existe un mot u tel que $s = \delta^*_{\mathcal{B}}(i_{\mathcal{B}},u)$.
On a alors $\varphi\big(\delta^*_{\mathcal{A}}(i_{\mathcal{A}},u)\big) = \delta^*_{\mathcal{B}}\big(\varphi(i_{\mathcal{A}},u)\big) = \delta^*_{\mathcal{B}}(i_{\mathcal{B}},u) = s$; en effet, la preuve faite en question **13** n'utilise pas la surjectivité. On obtient ainsi que φ est bien surjective.

On n'a pas utilisé l'accessibilité de $\mathcal{A}$.

Question 17
Pour définir une application d'un automate $\mathcal{A}$ vers un automate $\mathcal{B}$ on va parcourir (en profondeur depuis l'état initial) l'automate $\mathcal{A}$. Si $\mathcal{A}$ est accessible, on verra alors toutes les transitions. Si $\varphi(s) = s'$, $\delta_{\mathcal{A}}(s,x) = t$ et $\delta_{\mathcal{B}}(s',x) = t'$ on devra avoir $\varphi(t) = t'$. Ainsi chaque transition permet soit de définir l'image d'un sommet, si $\varphi(t)$ n'est pas encore défini, soit de vérifier que $\varphi(t)$ est bien l'extrémité de la transition appliquée dans $\mathcal{B}$. Si ce n'est pas le cas, il ne saurait exister de morphisme. On doit aussi vérifier la stabilité des états finaux.
On commence, biensûr en $i_{\mathcal{A}}$ dont l'image doit être $i_{\mathcal{B}}$
L'accessibilité de $\mathcal{B}$ permet de conclure la surjectivité.
On peut noter que cet algorithme prouve l'unicité d'un morphisme quand $\mathcal{A}$ est accessible.

Pour simplifier l'écriture, un test d'équivalence :

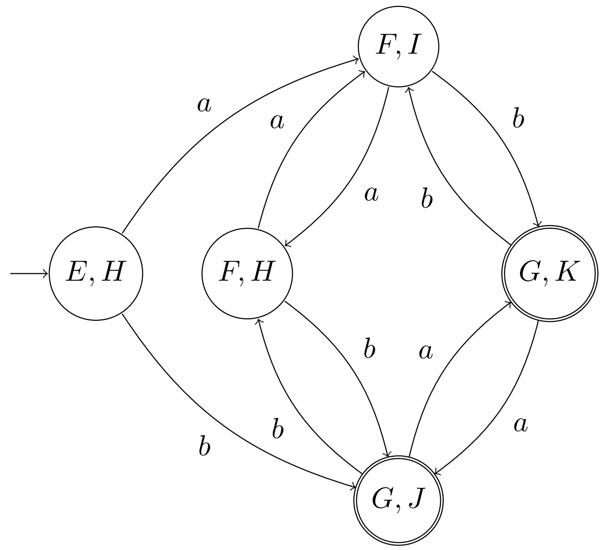
```
let ssi b1 b2 = (b1 && b2) || (not b1 && not b2);;
```

```
let existe_morphisme q1 q2 =
  let (n1, delta1, f1) = q1 in
  let (n2, delta2, f2) = q2 in
  let vus = Array.make n1 false in
  let phi = Array.make n1 (-1) in
  let existe = ref true in
  let traiter s t =
    if phi.(s) = -1
    then begin
         if ssi f1.(s) f2.(t)
         then phi.(s) <- t
         else existe := false end
    else if phi.(s) <> t
         then existe := false in
  let rec aux s =
    if not vus.(s)
    then begin
         vus.(s) <- true;
         let t = phi.(s) in
         let sa, sb = delta1.(s) in
         let ta, tb = delta2.(t) in
         traiter sa ta;
         traiter sb tb;
         if !existe then (aux sa; aux sb) end in
  phi.(0) <- 0;
  aux 0;
  !existe, phi;;
```

4 Constructions de morphismes

4.1 Automate produit

Question 18
On calcule les états accessibles pas-à-pas depuis l'état initial (E, H).



Question 19
On numérote les sommets par $k_1 + n_1.k_2$ où $k - 1$ et k_2 sont les indices respectivement du premier et du second automate et n_1 est la taille du premier.

```
let produit q1 q2 =
  let (n1, delta1, f1) = q1 in
  let (n2, delta2, f2) = q2 in
  let n = n1*n2 in
  let delta = Array.make n (0,0) in
  let f = Array.make n false in
  for k1 = 0 to (n1 - 1) do
    for k2 = 0 to (n2 - 1) do
      let k = k1 + k2*n1 in
      let (a1, b1) = delta1.(k1) in
      let (a2, b2) = delta2.(k2) in
      delta.(k) <- (a1 + a2*n1, b1 + b2*n1);
      f.(k) <- f1.(k1) && f2.(k2) done done;
  (n, delta, f);;
```

Question 20
De la même manière qu'à la question **13** on prouve, par récurrence sur $|u|$ que

$$\delta^*_{\mathcal{A}\times\mathcal{A}'}\big((s,s'),u\big) = \big(\delta^*_{\mathcal{A}}(s,u), \delta^*_{\mathcal{A}'}(s',u)\big)$$

Si (s,s') est accessible, on peut écrire $(s,s') = \delta^*_{\mathcal{A}\times\mathcal{A}'}\big((i_{\mathcal{A}}, i_{\mathcal{A}'}),u\big) = \big(\delta^*_{\mathcal{A}}(i_{\mathcal{A}},u), \delta^*_{\mathcal{A}'}(i_{\mathcal{A}},u)\big)$.
u est reconnu par $\mathcal{A}$ si et seulement s'il est reconnu par $\mathcal{A}'$ d'où $q \in F_{\mathcal{A}} \iff q' \in F_{\mathcal{A}'}$.

Question 21
On considère la projection de $\mathcal{A} \times \mathcal{A}'$ sur $\mathcal{A}$: $(s,s') \mapsto s$ et on note φ_1 sa restriction à l'ensemble des états accessibles de $\mathcal{A} \times \mathcal{A}'$.

1. Tout état s de $\mathcal{A}$ est accessible donc il existe u tel que $s = \delta^*_{\mathcal{A}}(i_{\mathcal{A}},u)$. Si on note $s' = \delta^*_{\mathcal{A}'}(i_{\mathcal{A}},u)\big)$, on a (s,s') accessible dans $\mathcal{A} \times \mathcal{A}'$ donc $s = \varphi_1(s,s')$. φ_1 est surjective.
2. $\varphi_1(i_{\mathcal{A}\times\mathcal{A}'}) = \varphi_1(i_{\mathcal{A}}, i_{\mathcal{A}'}) = i_{\mathcal{A}}$.
3. $\varphi_1\big(\delta_{\mathcal{A}\times\mathcal{A}'}(('s,s'),x)\big) = \varphi_1\big(\delta_{\mathcal{A}}(s,x), \delta_{\mathcal{A}'}(s',x)\big) = \delta_{\mathcal{A}}(s,x) = \delta_{\mathcal{A}}\big(\varphi_1(s,s'),x\big)$.
4. Si $(s,s') \in F_{\mathcal{A}\times\mathcal{A}'} = F_{\mathcal{A}} \times F_{\mathcal{A}'}$ alors $\varphi_1(s,s') = s \in F_{\mathcal{A}}$.
   Inversement, pour $s \in F_{\mathcal{A}}$, il existe $s' \in F_{\mathcal{A}'}$ tel que (s,s') est accessible dans $\mathcal{A}\times\mathcal{A}'$. La question précédente montre que s' est final dans $\mathcal{A}'$ d'où $(s,s') \in F_{\mathcal{A}\times\mathcal{A}'}$. Ainsi $\varphi_1(s,s') \in F_{\mathcal{A}}$ implique $(s,s') \in F_{\mathcal{A}\times\mathcal{A}'}$. On en conclut l'équivalence.

Les 4 propriétés prouve que φ_1 est un morphisme d'automates de la partie accessible de $\mathcal{A} \times \mathcal{A}'$ vers $\mathcal{A}$. De même la restriction de la seconde projection est un morphisme d'automates de la partie accessible de $\mathcal{A} \times \mathcal{A}'$ vers $\mathcal{A}'$.

4.2 Diagramme d'automates

Question 22
On nomme **chemin** de p vers q une suite d'états vérifiant les propriétés de la définition.
- (p) est un chemin de p vers p donc $p \equiv p$.
- Si $p \equiv q$ on considère un chemin $(p, s_1, \ldots, s_{k-1}, q)$ de p vers q.
  $(q, s_{k-1}, \ldots, s_1, p)$ est alors un chemin de q vers p : $p \equiv q \Longrightarrow q \equiv p$.
- Si $p \equiv q$ et $q \equiv r$ on considère un chemin $(p, s_1, \ldots, s_{k-1}, q)$ de p vers q et un chemin $(q, t_1, \ldots, t_{l-1}, r)$ de q vers r.
  $(p, s_1, \ldots, s_{k-1}, q, t_1, \ldots, t_{l-1}, r)$ est alors un chemin de p vers r : $p \equiv q \wedge q \equiv r \Longrightarrow q \equiv r$.

$\equiv$ est une relation d'équivalence.

Question 23
On considère un chemin $(s_0, s_1, \ldots, s_{k-1}, s_k)$ avec $s_0 = p$ et $s_k = q$ pour $p \equiv q$.
Si on a $\varphi(s_i) = \varphi(s_{i+1})$ alors $\varphi\big(\delta_{\mathcal{B}}(s_i, x)\big) = \delta_{\mathcal{A}}\big(\varphi(s_i), x\big) = \delta_{\mathcal{A}}\big(\varphi(s_{i+1}), x\big) = \varphi\big(\delta_{\mathcal{B}}(s_{i+1}, x)\big)$.
De même pour ψ.
Ainsi, si on note $t_i = \delta_{\mathcal{B}}(s_i, x)$, on a toujours $\varphi(t_i) = \varphi(t_{i+1})$ ou $\psi(t_i) = \psi(t_{i+1})$ donc
$(t_0, t_1, \ldots, t_{k-1}, t_k)$ est un chemin de $\delta_{\mathcal{B}}(p, x)$ vers $\delta_{\mathcal{B}}(q, x)$. $p \equiv q \Longrightarrow \delta_{\mathcal{B}}(p, x) \equiv \delta_{\mathcal{B}}(q, x)$.

Question 24
On suppose $p \equiv q$ et on considère un chemin $(s_0, s_1, \ldots, s_{k-1}, s_k)$ avec $s_0 = p$ et $s_k = q$.
Montrons que $s_i \in F_{\mathcal{B}} \Longrightarrow s_{i+1} \in F_{\mathcal{B}}$ pour $i < k$.
- Si on a $\varphi(s_i) = \varphi(s_{i+1})$, on sait que $s_i \in F_{\mathcal{B}}$ donc $\varphi(s_i) \in F_{\mathcal{A}}$ d'où $\varphi(s_{i+1}) \in F_{\mathcal{A}}$
  puis $s_{i+1} \in F_{\mathcal{B}}$ d'après la propriété (3) d'un morphisme.
- De même si on a $\psi(s_i) = \psi(s_{i+1})$.

On en déduit que si $s_0 = p \in F_{\mathcal{B}}$ alors $s_i \in F_{\mathcal{B}}$ pour tout i donc $q = s_k \in F_{\mathcal{B}}$.
De même, par symétrie de $\equiv$, $p \equiv q$ et q final impliquent p final.

Question 25
- L'ensemble des sommets est $\mathcal{C} = \{S_0, S_1, \ldots, S_{\ell-1}\}$.
- La question **23** montre que si on a $[s] = [t]$ alors $[\delta_{\mathcal{B}}(s, x)] = [\delta_{\mathcal{B}}(t, x)]$.
  On peut donc définir $\delta_{\mathcal{C}}([s], x) = [\delta_{\mathcal{B}}(s, x)]$.
- La question **24** montre que, pour $[s] = [t]$, $s \in F_{\mathcal{B}} \Longrightarrow t \in F_{\mathcal{B}}$. La propriété $s \in F_{\mathcal{B}}$ ne dépend donc pas du représentant de $[s]$ et on peut définir sans ambiguïté $F_{\mathcal{C}}$ comme l'ensemble des classes d'équivalence des éléments de $F_{\mathcal{B}}$.

Ainsi $\langle \mathcal{C}, [i_{\mathcal{B}}], \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle$ est un automate.

1. Par construction $\eta : s \mapsto [s]$ est surjective.
2. $\eta(i_{\mathcal{B}}) = [i_{\mathcal{B}}] = i_{\mathcal{C}}$.
3. $\eta\big(\delta_{\mathcal{B}}(s, x)\big) = [\delta_{\mathcal{B}}(s, x)] = \delta_{\mathcal{C}}([s], x) = \delta_{\mathcal{C}}\big(\eta(s), x\big)$.
4. On a vu que $\eta(s) = [s] \in F_{\mathcal{C}}$ équivaut à $s \in F_{\mathcal{B}}$.

Ainsi η est un morphisme d'automates de $\mathcal{B}$ vers $\mathcal{C}$.

Question 26
On veut $\eta = \varphi' \circ \varphi$, donc on doit définir $\varphi'(t) = [s]$ pour $t = \varphi(s)$.
Il reste à prouver que φ' est bien défini. Or $t = \varphi(s) = \varphi(s')$ implique que $s \equiv s'$ donc $[s] = [s']$: la définition de $\varphi'(t)$ ne dépend pas de l'antécédent choisi. De plus φ est surjective donc on peut définir $\varphi'(t)$ pour tout état $t \in \mathcal{A}$.

Pour $t \in \mathcal{A}$ on considère $s \in \mathcal{B}$ tel que $t = \varphi(s)$ dans les deux derniers items.

On a alors $\varphi'(t) = \varphi' \circ \varphi(s) = \eta(s)$.

1. Pour tout état $[s]$ de $\mathcal{C}$, $[s] = \varphi'(t)$ avec $t = \varphi(s)$: φ' est surjective.
2. $i_{\mathcal{A}} = \varphi(i_{\mathcal{B}})$ donc $\varphi'(i_{\mathcal{A}}) = [i_{\mathcal{B}}] = i_{\mathcal{C}}$.
3. $\varphi'\big(\delta_{\mathcal{A}}(t,x)\big) = \varphi'\big(\delta_{\mathcal{A}}(\varphi(s),x)\big) = \varphi' \circ \varphi\big(\delta_{\mathcal{A}}(s,x)\big) = \eta\big(\delta_{\mathcal{A}}(s,x)\big) = \delta_{\mathcal{C}}\big(\eta(s),x\big) = \delta_{\mathcal{C}}\big(\varphi'(t),x\big)$.
4. $t = \varphi(s) \in F_{\mathcal{A}} \iff s \in F_{\mathcal{B}} \iff \eta(s) \in F_{\mathcal{C}} \iff \varphi'(t) \in F_{\mathcal{C}}$.

φ' est un morphisme d'automates de $\mathcal{A}$ vers $\mathcal{C}$.

De même on définit un morphisme d'automates de $\mathcal{A}'$ vers $\mathcal{C}$ par $\psi'(t) = [s]$ pour $t = \psi(s)$.

Question 27

Pour obtenir une complexité linéaire on va maintenir un tableau de substitution des valeurs. Quand on lit une valeur

— soit elle a déjà été rencontrée et sa valeur de substitution est connue, on l'applique

— soit elle est nouvelle, on définit sa substitution en incrémentant un compteur et on l'applique.

On doit faire une première lecture du tableau pour déterminer la valeur maximale. On suppose le tableau non vide.

```
let maxi_tableau t =
  let n = Array.length t in
  let maxi = ref t.(0) in
  for i = 1 to (n-1) do
    if t.(i) > !maxi then maxi := t.(i) done;
  !maxi;;
```

```
let renomme t =
  let n = Array.length t in
  let maxi = maxi_tableau t in
  let code = Array.make (maxi + 1) (-1) in
  let out = Array.make n (-1) in
  let subs = ref 0 in
  for i = 0 to (n-1) do
    let k = t.(i) in
    match code.(k) with
    |-1 -> code.(k) <- !subs;
           out.(i) <- !subs;
           incr subs
    |p -> out.(i) <- p done;
  !out;;
```

La complexité est linéaire.

Question 28

La structure naturelle pour gérer une relation d'équivalence est la structure **union-find** telle qu'utilisée dans l'algorithme de Kruskal.

```
let creerUF n =
  Array.init n (fun i -> i);;

let rec find u i =
  if u.(i) <> i then
  find u (u.(i))
  else i;;

let union u i j =
  let k = find u i in
  let l = find u j in
  if k <> l then u.(k) <- l;;
```

On crée donc les classes d'équivalences par adjonction des classes des paires d'éléments qui ont une image en commun. Il faudra ensuite remplacer, dans le tableau, les antécédents par la racine des arbres.

```
let relation phi psi =
  let n = Array.length phi in
  let uf = creerUF n in
  for i = 0 to (n - 2) do
    for j = (i + 1) to (n - 1) do
      if phi.(i) = phi.(j) || psi.(i) = psi.(j)
      then union uf i j done done;
  for k = 0 to (n-1) do uf.(k) <- find uf k done;
  renomme uf;;
```

5 Réductions d'automates

5.1 Existence et unicité

Question 29
On définit $\mathcal{B}$ comme la partie accessible de $\mathcal{A} \times \mathcal{A}'$, la question **21** montre qu'il existe des morphismes φ et ψ respectivement de $\mathcal{B}$ vers $\mathcal{A}$ et de $\mathcal{B}$ vers $\mathcal{A}'$.
Les questions **25** et **26** permettent alors de définir un automate $\mathcal{C}$ et deux morphismes φ' et ψ' respectivement de $\mathcal{A}$ vers $\mathcal{C}$ et de $\mathcal{A}'$ vers $\mathcal{C}$.

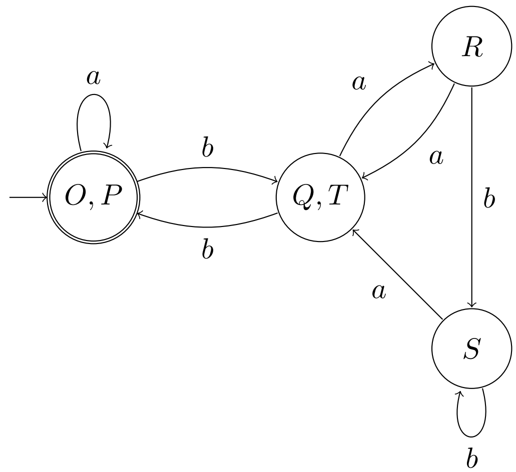
Question 30
Dans la partie accessible de $\mathcal{A}_3 \times \mathcal{A}_4$, on a $(E, H) \equiv (F, H) \equiv (F, I)$ et $(G, J) \equiv (G, K)$.
0 représente la classe de (E, H), c'est l'état initial, 1 est la classe de (G, J), c'est le seul état final.
L'automate $\mathcal{C}$ est isomorphe à l'automate $\mathcal{A}_2$.
On a alors $\varphi'(E) = \varphi'(F) = 0$, $\varphi'(H) = 1$, $\psi'(H) = \psi'(I) = 0$ et $\psi'(J) = \psi'(K) = 1$.

Question 31
$\mathcal{A}$ et $\mathcal{A}'$ sont deux automates reconnaissant le même langage L et de cardinal minimal dans $\mathfrak{K}_L$.
Dans la construction de la question **29** on définit un automate $\mathcal{C}$ et deux morphismes φ' et ψ' respectivement de $\mathcal{A}$ vers $\mathcal{C}$ et de $\mathcal{A}'$ vers $\mathcal{C}$.
D'après la question **13**, $\mathcal{C}$ reconnaît aussi L. Son cardinal est donc supérieur ou égal à celui de $\mathcal{A}$ et de $\mathcal{A}'$; comme φ' et ψ' sont surjectifs, le cardinal de $\mathcal{C}$ est inférieur ou égal à celui de $\mathcal{A}$ et de $\mathcal{A}'$. On en déduit que les 3 automates ont même cardinal.
D'après la question **14** on peut conclure que φ' est un isomorphisme de $\mathcal{A}$ vers $\mathcal{C}$ et ψ' un isomorphisme de $\mathcal{A}'$ vers $\mathcal{C}$ donc $\psi'^{-1} \circ \varphi$ est un isomorphisme de $\mathcal{A}$ vers $\mathcal{A}'$.

Question 32
C'est la même démonstration, sans supposer que $\mathcal{A}'$ est de cardinal minimal.
On prouve encore que φ' est un isomorphisme de $\mathcal{A}$ vers $\mathcal{C}$.
Dans ce cas $\varphi'^{-1} \circ \psi'$ est un morphisme de $\mathcal{A}'$ vers $\mathcal{A}$.

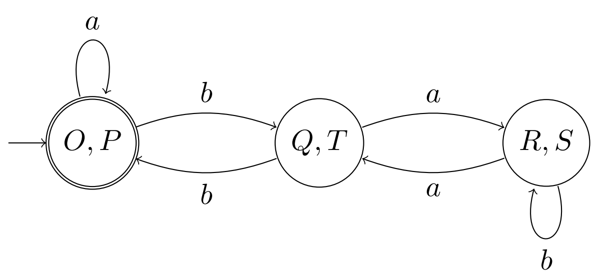
5.2 Construction

Question 33
Pour fusionner l'automate en assimilant les états O et P, il faut aussi fusionner les états Q et T pour que l'image de l'état (O, P) par la transition b puisse être définie. Le morphisme est représenté par les antécédents de chaque états de $\mathcal{A}_6^{O,P}$.



Question 34
Il n'est pas possible de fusionner Q et R car $\delta(Q,b) = O$ et $\delta(R,b) = S$ l'un des état est final et l'autre non ; la condition (4) interdit alors la possibilité d'un morphisme ψ tel que $\psi(O) = \psi(S)$. Or si $\psi(Q) = \psi(R)$, alors $\psi(O) = \psi(\delta(Q,b)) = \delta'(\psi(Q),b) = \delta'(\psi(R),b) = \psi(\delta(R,b)) = \psi(S)$.

Question 35
On peut encore fusionner R et S.



Question 36
On commence par créer le graphe avec une matrice des listes d'adjacence.

```
let graphe q =
  let n, delta, f = q in
  let adj = Array.make_matrix n n [] in
  for p = 0 to (n-1) do
    for q = 0 to (n-1) do
      let pa, pb = delta.(p) in
      let qa, qb = delta.(q) in
      adj.(pa).(qa) <- (p, q) :: adj.(pa).(qa);
      adj.(pb).(qb) <- (p, q) :: adj.(pb).(qb) done done;
  adj;;
```

On effectue alors un parcours du graphe avec le tableau.

```
let table_de_predecesseurs q =
  let (n, delta, f) = q in
  let g = graphe q in
  let pred = Array.make_matrix n n false in
  let rec visiter (p, q) =
    if not pred.(p).(q)
    then begin pred.(p).(q) <- true;
         List.iter visiter g.(p).(q) end in
  for p = 0 to (n-1) do
```

```
    for q = 0 to (n-1) do
      if f.(p) && (not f.(q)) || (not f.(p)) && f.(q)
      then visiter p q done done;
  pred;;
```

Question 37

Si deux états p et q d'un automate $\mathcal{A}$ sont marqués à `false` dans le tableau des prédécesseurs, cela signifie que, pour tout mot u, $\delta^*(p,u)$ et $\delta^*(q,u)$ sont finaux en même temps. On note $\equiv_F$ cette relation, il est facile de montrer que c'est une relation d'équivalence.
Si on a $p \equiv_F q$ alors on doit avoir $\delta(p,x) \equiv_F \delta(q,x)$ car sinon un mot u enverrait $\delta(p,x)$ dans F et $\delta(q,x)$ dans $Q \setminus F$ (ou l'inverse) et le mot xu aurait la même propriété pour p et q ce qui est exclu.
L'action du mot vide montre que, si $p \equiv_F q$ alors p et q sont dans F en même temps.

On peut ainsi définir un automate $\mathcal{A}_0$ à partir des classes d'équivalence, comme à la question **25**, ainsi qu'un morphisme de $\mathcal{A}$ vers $\mathcal{A}_0$. On a vu que $\mathcal{A}_0$ reconnaît le même langage que $\mathcal{A}$.
Dans $\mathcal{A}_0$, deux états distincts peuvent être séparés par un mot : pour $[p] \neq [q]$ on a (p,q) marqué comme `true` dans la table des prédécesseurs donc il existe u tel que $\delta^*(p,u) \in F$ et $\delta^*(q,u) \notin F$ (ou l'inverse). S'il existe un morphisme ψ' de $\mathcal{A}_0$ vers $\mathcal{A}'$ alors on ne peut avoir $\psi'([p]) = \psi'([q])$ car le transporté par u devrait être à la fois final et non final en raison des propriétés d'un morphisme.
En particulier le morphisme de $\mathcal{A}_0$ vers un automate minimal est injectif donc c'est un isomorphisme ; on en déduit que $\mathcal{A}_0$ est minimal.
On commence par la définition des classes d'équivalences. La fonction renvoie deux tableaux :

- un tableau donnant le numéro de classe pour chaque état
- et tableau donnant un représentant de chaque classe.

```
let equivalence q =
  let n, delta, f = q in
  let pred = table_de_predecesseurs q in
  let classe = Array.make n (-1) in
  let compteur = ref 0 in
  for p = 0 to (n-1) do
    if classe.(p) = -1
    then begin classe.(p) <- !compteur;
         for q = (p+1) to (n-1) do
           if not pred.(p).(q)
           then classe.(q) <- !compteur done;
         incr compteur end
    done;
  let repr = Array.make !compteur (-1) in
  for p = 0 to (n-1) do repr.(classe.(p)) <- p done;
  classe, repr;;
```

```
let reduit q =
  let n, delta, f = q in
  let classe, repr = equivalence q in
  let p = Array.length repr in
  let delta1 = Array.make p (0,0) in
  let f1 = Array.make p false in
  for i = 0 to (p-1) do
    let sa, sb = delta.(repr.(i)) in
    delta1.(i) <- (classe.(sa), classe.(sb));
    f1.(i) <- f.(repr.(i)) done;
  (p, delta1, f1);;
```