COMPOSITION D'INFORMATIQUE n°2

Corrigé

⋆ ⋆ ⋆

1. $L(\mathcal{A}_1)$ est l'ensemble des mots de longueur impaire.
2. $L(\mathcal{A}_2)$ est l'ensemble des mots contenant un nombre impair de b.
3. L'expression $((a+b)(a+b))^*(a+b)$ convient.
4. L'expression $a^*b(a^*ba^*ba^*)^*$ convient.
5. On peut écrire :

```
let a2 =  {delta = [|(0, 1); (1, 0)|]; finals = [|false; true|]};;
```

6. On commence par créer le tableau, puis on écrit une fonction auxiliaire qui parcourt les éléments de la liste, en gardant également en mémoire l'indice `i` où on se trouve dans la liste. Pour chaque élément, on modifie le tableau `t` en conséquence (notons ici que les indices contenus dans `t` seront les dernières occurrences).

```
let numero n lst =
    let t = Array.make n (-1) in
    let rec parcours i = function
        | []     -> ()
        | x :: q -> t.(x) <- i; parcours (i + 1) q in
    parcours 0 lst;
    t;;
```

   En remarque : on aurait pu remplacer la fonction `parcours` par un appel à `List.iteri` bien choisi.

7. On écrit l'algorithme de parcours de graphe. Notons que chaque sommet a au plus deux voisins. La fonction `parcours` prend en argument un sommet, le traite s'il n'a pas été vu et relance un parcours depuis ses deux voisins. Comme on utilise ici une référence de liste pour garder en mémoire les états accessibles, il faut penser à la retourner une fois le parcours terminé.

```
let etats_accessibles aut =
    let n = Array.length aut.finals in
    let vus = Array.make n false in
    let etats = ref [] in
    let rec parcours x =
        if not vus.(x) then begin
           vus.(x) <- true;
           etats := x :: !etats;
           let y, z = aut.delta.(x) in
           parcours y; parcours z
        end in
    parcours 0;
    List.rev !etats;;
```

8. On propose le code :

```
let partie_accessible aut =
    let n = Array.length aut.finals in
    let lst = etats_accessibles aut in
    let t = numero n lst in
    let n' = List.length lst in
    let delta' = Array.make n' (0, 0) in
    let finals' = Array.make n' false in
    for x = 0 to n - 1 do
        if t.(x) >= 0 then begin
           let (y, z) = aut.delta.(x) in
           finals'.(t.(x)) <- aut.finals.(x);
           delta'.(t.(x)) <- (t.(y), t.(z));
        end
    done;
    {finals = finals'; delta = delta'};;
```

Comme la liste des états accessibles ne contient pas de doublons, l'appel à `numero` permet de renuméroter correctement les sommets (l'état initial reste le sommet 0). Pour créer le nouvel automate, on commence par déterminer n', créer les nouveaux tableaux des transitions et des états finaux, puis on parcourt tous les sommets pour effectuer la renumérotation. On se sert du tableau `t` en guise d'intermédiaire.

9. On peut compléter de la manière suivante :

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

C'est ici la seule manière de respecter la règle (4). On constate que les autres règles sont bien conservées (la règle (3) est la plus longue à vérifier).

10. On peut donner le morphisme suivant :

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

À nouveau, c'est la seule manière de respecter la règle (4).

11. Pour respecter la règle (4), le seul morphisme φ qui conviendrait vérifierait $\varphi(A) = C$ et $\varphi(B) = D$. Dans ce cas, les règles (1) et (2) sont respectées, mais on constate que $\varphi(\delta(A, a)) = \varphi(B) = D \neq \delta(\varphi(A), a) = \delta(C, a) = C$. La règle (3) n'est pas respectée.

12. À nouveau, pour respecter la règle (4), il faut $\varphi(L) = \varphi(N) = C$ et $\varphi(M) = D$. On constate cependant que $\varphi(\delta(N, a)) = \varphi(M) = D \neq \delta(\varphi(N), a) = \delta(C, a) = C$. La règle (3) n'est pas respectée.

13. Soit $\mathcal{A}$ et $\mathcal{B}$ deux automates tels qu'il existe un morphisme d'automate φ de A vers B. Montrons par induction que pour tout $q \in Q_{\mathcal{A}}$ et tout $u \in \Sigma^*$, $\varphi(\delta^*_{\mathcal{A}}(q, u)) = \delta^*_{\mathcal{B}}(\varphi(q), u)$:
    - si $u = \varepsilon$, alors $\varphi(\delta^*_{\mathcal{A}}(q, u)) = \varphi(q) = \delta^*_{\mathcal{B}}(\varphi(q), u)$ (par définition des fonctions de transition étendues) ;
    - supposons le résultat vrai pour $u \in \Sigma^*$ fixé, et soit $\sigma \in \Sigma$. Alors :

$$\varphi(\delta^*_{\mathcal{A}}(q, u\sigma)) = \varphi(\delta_{\mathcal{A}}(\delta^*_{\mathcal{A}}(q, u), \sigma)) = \delta_{\mathcal{B}}(\varphi(\delta^*_{\mathcal{A}}(q, u)), \sigma) = \delta_{\mathcal{B}}(\delta^*_{\mathcal{B}}(\varphi(q), u), \sigma) = \delta^*_{\mathcal{B}}(\varphi(q), u\sigma)$$

Le résultat est donc prouvé par induction. Dès lors, pour $u \in \Sigma^*$, on a :

$$\begin{array}{lll} u \in L(\mathcal{A}) & \Leftrightarrow & \delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u) \in F_{\mathcal{A}} \\ & \Leftrightarrow & \varphi(\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u)) \in F_{\mathcal{B}} \quad \text{par règle (4)} \\ & \Leftrightarrow & \delta^*_{\mathcal{B}}(\varphi(i_{\mathcal{A}}), u) \in F_{\mathcal{B}} \quad \text{par l'égalité prouvée plus haut} \\ & \Leftrightarrow & \delta^*_{\mathcal{B}}(i_{\mathcal{B}}, u) \in F_{\mathcal{B}} \quad \text{par règle (2)} \\ & \Leftrightarrow & u \in L(\mathcal{B}) \end{array}$$

14. Si un morphisme φ est entre deux automates ayant le même nombre d'états n, alors comme φ est surjective (règle (1)) et que n est fini, φ est nécessairement bijective. Si c'est le cas, on voit facilement que φ^{-1} vérifie les règles (1), (2) et (4). Vérifions plus en détail la règle (3) : soit $q \in \mathbb{Q}_{\mathcal{B}}$ et $\sigma \in \Sigma$. Alors :

$$\varphi(\delta_{\mathcal{A}}(\varphi^{-1}(q), \sigma)) = \delta_{\mathcal{B}}(\varphi(\varphi^{-1}(q)), \sigma) = \delta_{\mathcal{B}}(q, \sigma)$$

On en déduit que $\varphi^{-1}(\delta_{\mathcal{B}}(q,\sigma)) = \delta_{\mathcal{A}}(\varphi^{-1}(q),\sigma)$ ce qui assure la règle (3).

15. Soit $\mathcal{A}$, $\mathcal{B}$ et $\mathcal{C}$ trois automates, et φ_1 et φ_2 des morphismes d'automates respectivement de $\mathcal{A}$ vers $\mathcal{B}$ et de $\mathcal{B}$ vers $\mathcal{C}$. Montrons que $\varphi_2 \circ \varphi_1$ est un morphisme de $\mathcal{A}$ vers $\mathcal{C}$:
    - φ_1 et φ_2 sont surjectives donc $\varphi_2 \circ \varphi_1$ est surjective ;
    - $\varphi_2 \circ \varphi_1(i_{\mathcal{A}}) = \varphi_2(i_{\mathcal{B}}) = i_{\mathcal{C}}$;
    - $\forall q \in Q_{\mathcal{A}}, \forall \sigma \in \Sigma, \varphi_2 \circ \varphi_1(\delta_{\mathcal{A}}(q,\sigma)) = \varphi_2(\delta_{\mathcal{B}}(\varphi_1(q),\sigma)) = \delta_{\mathcal{C}}(\varphi_2 \circ \varphi_1(q),\sigma))$;
    - $\forall q \in Q_{\mathcal{A}}, q \in F_{\mathcal{A}} \Leftrightarrow \varphi_1(q) \in F_{\mathcal{B}} \Leftrightarrow \varphi_2 \circ \varphi_1(q) \in F_{\mathcal{C}}$.

    Les quatre règles sont bien vérifiées.

16. Supposons $\mathcal{A}$ et $\mathcal{B}$ deux automates accessibles et φ une application de $Q_{\mathcal{A}}$ et $Q_{\mathcal{B}}$ vérifiant les règles (2), (3) et (4). Montrons que φ est surjective : soit $q \in Q_{\mathcal{B}}$. $\mathcal{B}$ étant accessible, il existe un mot $u \in \Sigma^*$ tel que $\delta^*_{\mathcal{B}}(i_{\mathcal{B}}, u) = q = \delta^*_{\mathcal{B}}(\varphi(i_{\mathcal{A}}), u)$. Par la preuve du début de la question 13, on a $\delta^*_{\mathcal{B}}(\varphi(i_{\mathcal{A}}), u) = \varphi(\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u))$. On en déduit qu'il existe $p = \delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u) \in Q_{\mathcal{A}}$ (l'automate est complet) tel que $q = \varphi(p)$, ce qui assure la surjectivité. Notons que la règle (4) n'a pas été utilisée pour faire cette preuve, ni l'accessibilité de $\mathcal{A}$.

17. Notons que comme les deux automates sont accessibles, il ne peut exister qu'au plus un seul morphisme (les images sont déterminées de manière unique en utilisant la règle (3)). L'idée est donc d'effectuer un parcours de graphe et de remplir les images au fur et à mesure.

```
let trouver_morphisme aut1 aut2 =
    let n1 = Array.length aut1.finals in
    let phi = Array.make n1 (-1) in
    phi.(0) <- 0;
    let rec parcours q =
        let (qa, qb) = aut1.delta.(q) in
        let p = phi.(q) in
        let (pa, pb) = aut2.delta.(p) in
        verif qa pa && verif qb pb
    and verif q p =
        if phi.(q) = -1 then (phi.(q) <- p; aut1.finals.(q) = aut2.finals.(p) && parcours q)
        else phi.(q) = p in
    if parcours 0 then Some phi else None;;
```

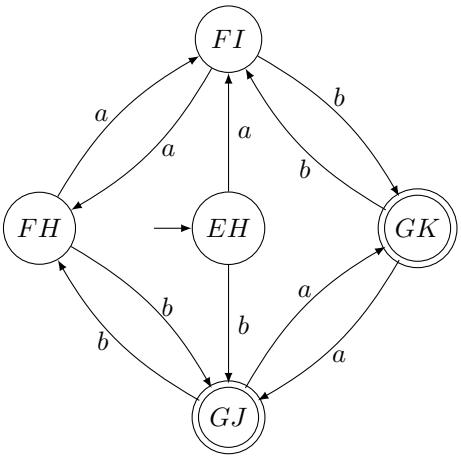
    La fonction précédente pose $\varphi(0) = 0$ et utilise deux fonctions auxiliaires récursives :
    - `parcours` prend en argument un état q du premier automate dont l'image p par φ a déjà été déterminée et renvoie un booléen. Pour ce faire, on détermine $\delta_1(q,a)$ et $\delta_2(p,a)$ (et de même pour b) et on vérifie qu'il n'y a pas d'incohérence sur chacune des paires avec la fonction `verif` ;
    - `verif` prend en argument un état q du premier automate et un état p du deuxième automate et agit selon les cas :
        * si $\varphi(q)$ n'a pas encore été déterminée, on pose $\varphi(q) = p$, on vérifie que q et p sont finaux ou non finaux simultanément et on relance un parcours depuis q ;
        * si $\varphi(q) \neq p$, alors il y a contradiction et la règle (3) ne peut pas être vérifiée : on arrête les calculs et on renvoie `false` ;
        * si $\varphi(q) = p$, alors on est revenu sur q par un cycle et il n'y a pas de contradiction. On ne relance pas de parcours et on renvoie `true`.

    Notons que grâce à la question 16, il n'y a pas besoin de vérifier la surjectivité.

18. Pour ne garder que les états accessibles, on peut commencer par remplir une table de transition :

État	a	b
$\rightarrow EH$	FI	GJ
FI	FH	GK
$GJ \rightarrow$	GK	FH
FH	FI	GJ
$GK \rightarrow$	GJ	FI

    Ainsi, la partie accessible de $\mathcal{A}_3 \times \mathcal{A}_4$ est :



19. La difficulté ici est de trouver une renumérotation cohérente pour les états de l'automate produit. Pour deux automates de taille n_1 et n_2 et $p, q \in [\![0, n_1 - 1]\!] \times [\![0, n_2 - 1]\!]$, on peut numéroter l'état (p, q) par $n_2 \times p + q$. On vérifie que cela fournit bien une bijection de $[\![0, n_1 - 1]\!] \times [\![0, n_2 - 1]\!]$ vers $[\![0, n_1 n_2 - 1]\!]$. On remplit les transitions et les états finaux par une double boucle.

```
let produit aut1 aut2 =
    let n1 = Array.length aut1.finals and
        n2 = Array.length aut2.finals in
    let n = n1 * n2 in
    let delta = Array.make n (0, 0) and
        finals = Array.make n false in
    for p = 0 to n1 - 1 do
        for q = 0 to n2 - 1 do
            let pq = p * n2 + q in
            finals.(pq) <- aut1.finals.(p) && aut2.finals.(q);
            let (pa, pb) = aut1.delta.(p) and (qa, qb) = aut2.delta.(q) in
            delta.(pq) <- (pa * n2 + qa, pb * n2 + qb)
        done;
    done;
    {delta; finals};;
```

20. Une preuve rapide par induction, comme à la question 13, permet de montrer que $\forall (q, q') \in Q_{\mathcal{A}} \times Q_{\mathcal{A}'}, \forall u \in \Sigma^*$, $\delta^*_{\mathcal{A}\times\mathcal{A}'}((q, q'), u) = (\delta^*_{\mathcal{A}}(q, u), \delta^*_{\mathcal{A}'}(q', u))$. Dès lors, si (q, q') est un état accessible de l'automate produit, il est clair que q et q' sont des états accessibles dans leurs automates respectifs. De plus, soit $u \in \Sigma^*$ tel que $(q, q') = \delta^*_{\mathcal{A}\times\mathcal{A}'}((i_{\mathcal{A}}, i_{\mathcal{A}'}), u)$. On a $q = \delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u)$ et $q' = \delta^*_{\mathcal{A}'}(i_{\mathcal{A}'}, u)$ et on peut conclure :

$$q \in F_{\mathcal{A}} \Leftrightarrow \delta^*_{\mathcal{A}}(i_{\mathcal{A}}, u) \in F_{\mathcal{A}} \Leftrightarrow u \in L(\mathcal{A}) \Leftrightarrow u \in L(\mathcal{A}') \Leftrightarrow \delta^*_{\mathcal{A}'}(i_{\mathcal{A}'}, u) \in F_{\mathcal{A}'} \Leftrightarrow q' \in F_{\mathcal{A}'}$$

21. Soit $\mathcal{A}$ et $\mathcal{B}$ deux automates accessibles reconnaissant le même langage et $\mathcal{C}$ la partie accessible de $\mathcal{A} \times \mathcal{B}$. On propose de montrer que $\varphi : \begin{array}{ccc} Q_{\mathcal{C}} & \to & Q_{\mathcal{A}} \\ (q, q') & \mapsto & q \end{array}$ est un morphisme de $\mathcal{C}$ vers $\mathcal{A}$. Vérifions que les règles (2), (3) et (4) sont bien vérifiées (la (1) sera donnée par la question 16) :
    - $\varphi(i_{\mathcal{C}}) = \varphi(i_{\mathcal{A}}, i_{\mathcal{B}}) = i_{\mathcal{A}}$;
    - $\forall (q, q') \in Q_{\mathcal{C}}, \forall \sigma \in \Sigma, \varphi(\delta_{\mathcal{C}}((q, q'), \sigma)) = \varphi((\delta_{\mathcal{A}}(q, \sigma), \delta_{\mathcal{B}}(q', \sigma)) = \delta_{\mathcal{A}}(q, \sigma) = \delta_{\mathcal{A}}(\varphi(q, q'), \sigma)$;
    - $\forall (q, q') \in Q_{\mathcal{C}}, (q, q') \in F_{\mathcal{C}} \Leftrightarrow q = \varphi(q, q') \in F_{\mathcal{A}}$ (l'implication est par définition de $F_{\mathcal{A}\times\mathcal{B}}$, la réciproque par la question 20).

22.
    - Réflexivité : soit $p \in Q_{\mathcal{A}}$. Alors $\varphi(p) = \varphi(p)$, donc pour $k = 0$, la suite $p = q_0, q_1 = p$ convient pour montrer $p \equiv p$;
    - Symétrie : soit $(p, q) \in Q^2_{\mathcal{A}}$ tels que $p \equiv q$. Soit $k \in \mathbb{N}$ et $p = q_0, q_1, \ldots, q_k = q$ une suite d'états associée. Alors en posant, pour $j \in [\![0, k]\!]$, $q'_j = q_{k-j}$, on vérifie que pour $j \in [\![1, k]\!]$, $\varphi(q'_j) = \varphi(q_{k-j}) = \varphi(q_{k-j+1}) = \varphi(q'_{j-1})$ (ou de même avec φ'). La suite $q = q'_0, q'_1, \ldots, q'_k = p$ permet bien de conclure que $q \equiv p$;
    - Transitivité : soient $(p, q, r) \in Q^3_{\mathcal{A}}$ tels que $p \equiv q \equiv r$, $p = q_0, q_1, \ldots, q_k = q$ et $q = q'_0, q'_1, \ldots, q'_\ell = r$ des suites d'états associées. Alors la suite $p = q_0, q_1, \ldots, q_k, q'_1, \ldots, q'_\ell = r$ permet bien de montrer que $p \equiv r$.

La relation $\equiv$ est donc bien une relation d'équivalence.

23. Soient $(p,q) \in Q_{\mathcal{A}}^2$ et $\sigma \in \Sigma$. Soit $p = q_0, q_1, \ldots, q_k = q$ une suite caractérisant l'équivalence de p et q. On pose, pour $i \in [\![0,k]\!], p_i = \delta_{\mathcal{A}}(q_i,\sigma)$. Montrons que $p_0, p_1, \ldots, p_k$ caractérise l'équivalence de $\delta_{\mathcal{A}}(p,\sigma)$ et $\delta_{\mathcal{A}}(q,\sigma)$. En effet, soit $i \in [\![0,k-1]\!]$ tel que $\varphi(q_i) = \varphi(q_{i+1})$. Alors :

$$\varphi(p_i) = \varphi(\delta_{\mathcal{A}}(q_i,\sigma)) = \delta_{\mathcal{B}}(\varphi(q_i),\sigma) = \delta_{\mathcal{B}}(\varphi(q_{i+1}),\sigma) = \varphi(\delta_{\mathcal{A}}(q_{i+1},\sigma)) = \varphi(p_{i+1})$$

On raisonne de même dans l'automate $\mathcal{B}'$ si $\varphi'(q_i) = \varphi'(q_{i+1})$, ce qui nous assure bien l'équivalence cherchée.

24. Soient $(p,q) \in Q_{\mathcal{A}}^2$. Soit $p = q_0, q_1, \ldots, q_k = q$ une suite caractérisant l'équivalence de p et q. Pour $i \in [\![0,k-1]\!]$, on a :

$$q_i \in F_{\mathcal{A}} \Leftrightarrow \varphi(q_i) \in F_{\mathcal{B}} \vee \varphi'(q_i) \in F_{\mathcal{B}'} \Leftrightarrow \varphi(q_{i+1}) \in F_{\mathcal{B}} \vee \varphi'(q_{i+1}) \in F_{\mathcal{B}'} \Leftrightarrow q_{i+1} \in F_{\mathcal{A}}$$

Une récurrence rapide permet alors de conclure que $q_0 \in F_{\mathcal{A}} \Leftrightarrow q_k \in F_{\mathcal{A}}$ ce qui est bien le résultat attendu.

25. On pose $\mathcal{C} = (Q_{\mathcal{C}}, i_{\mathcal{C}}, \delta_{\mathcal{C}}, F_{\mathcal{C}})$ défini par :
    - $Q_{\mathcal{C}} = \{S_0, S_1, \ldots, S_{\ell-1}\}$;
    - $i_{\mathcal{C}} = S_0 = \eta(i_{\mathcal{A}})$;
    - $F_{\mathcal{C}} = \{S \in Q_{\mathcal{C}}, S \cap F_{\mathcal{A}} \neq \emptyset\}$;
    - $\forall S \in Q_{\mathcal{C}}, \forall \sigma \in \Sigma, \delta_{\mathcal{C}}(S,\sigma) = \eta(\delta_{\mathcal{A}}(q,\sigma))$ où $q \in S$.

    Remarquons que la définition de $\delta_{\mathcal{C}}$ est cohérente d'après la question 23 (les images ne dépendent pas du représentant choisi). Montrons que $\mathcal{C}$ est accessible et que η est bien un morphisme de $\mathcal{A}$ vers $\mathcal{C}$:
    - $\eta(i_{\mathcal{A}}) = S_0 = i_{\mathcal{C}}$ (la règle (2) est vérifiée) ;
    - $\forall q \in Q_{\mathcal{A}}, \forall \sigma \in \Sigma, \eta(\delta_{\mathcal{A}}(q,\sigma)) = \delta_{\mathcal{C}}(\eta(q),\sigma)$ par définition de $\delta_{\mathcal{C}}$ (en utilisant la question 23). La règle (3) est vérifiée ;
    - $q \in F_{\mathcal{A}} \Leftrightarrow \exists q' \in \eta(q), q' \in F_{\mathcal{A}} \Leftrightarrow \eta(q) \in F_{\mathcal{C}}$ (le sens retour de la première équivalence est donné par la question 24). La règle (4) est vérifiée ;
    - Finalement, on montre par une preuve similaire à la question 13 que pour $S \in Q_{\mathcal{C}}$ et $u \in \Sigma^*$, $\delta_{\mathcal{C}}^*(S,u) = \eta(\delta_{\mathcal{A}}^*(q,u)$ où $u \in S$. Comme $\mathcal{A}$ est accessible, on en déduit que pour $S \in Q_{\mathcal{C}}$, il existe $q \in S$ qui est accessible dans $\mathcal{A}$, donc $\exists u \in \Sigma^*, \delta_{\mathcal{A}}^*(i_{\mathcal{A}},u) = q$, et alors on a $\delta_{\mathcal{C}}^*(i_{\mathcal{C}},u) = S$. L'automate $\mathcal{C}$ est bien accessible.

    On conclut que η est bien un morphisme d'automates (la règle (1) est donnée par la question 16).

26. On définit ψ de la manière suivante : soit $q \in Q_{\mathcal{B}}$, il existe $p \in Q_{\mathcal{A}}$ tel que $\varphi(p) = q$ (car φ est surjective). On pose alors $\psi(q) = \eta(p)$. Dans un premier temps, ψ est bien définie, car si $\varphi(p) = \varphi(p')$, alors $p \equiv p'$ (donc $\psi(q)$ ne dépend pas du choix de l'antécédent de q par φ). Montrons que ψ est bien un morphisme de $\mathcal{B}$ vers $\mathcal{C}$:
    - $\varphi(i_{\mathcal{A}}) = i_{\mathcal{B}}$ donc $\psi(i_{\mathcal{B}}) = \eta(i_{\mathcal{A}}) = i_{\mathcal{C}}$: la règle (2) est vérifiée ;
    - $\forall q \in Q_{\mathcal{B}}, \forall \sigma \in \Sigma, \psi(\delta_{\mathcal{B}}(q,\sigma)) = \psi(\delta_{\mathcal{B}}(\varphi(p),\sigma))$ où $p \in Q_{\mathcal{A}}$ est un antécédent de q par φ. On en déduit $\psi(\delta_{\mathcal{B}}(q,\sigma)) = \psi(\varphi(\delta_{\mathcal{A}}(p,\sigma)) = \eta(\delta_{\mathcal{A}}(p,\sigma)) = \delta_{\mathcal{C}}(\eta(p),\sigma) = \delta_{\mathcal{C}}(\psi(q),\sigma)$ ce qui donne la règle (3) ;
    - $\forall q \in Q_{\mathcal{B}}, q \in F_{\mathcal{B}} \Leftrightarrow \varphi(p) \in F_{\mathcal{B}}$ où $p \in Q_{\mathcal{A}}$ est un antécédent de q par $\varphi \Leftrightarrow p \in F_{\mathcal{A}} \Leftrightarrow \eta(p) \in F_{\mathcal{C}} \Leftrightarrow \psi(q) \in F_{\mathcal{C}}$ d'où la règle (4).

    Comme $\mathcal{C}$ est accessible, ψ est bien un morphisme d'automate. On raisonne de même pour ψ'.

27. On propose le code suivant :

```
let maximum t = Array.fold_left max 0 t;;

let renomme t =
    let n = Array.length t in
    let corres = Array.make (maximum t + 1) (-1)
    and t' = Array.make n 0 and num = ref 0 in
    for i = 0 to n - 1 do
        if corres.(t.(i)) = -1 then
           (corres.(t.(i)) <- !num; incr num);
        t'.(i) <- corres.(t.(i))
    done;
    t';;
```

Dans un premier temps, on commence par déterminer la valeur maximale apparaissant dans le tableau (car c'est un majorant de ℓ, sachant que les valeurs sont distinctes). Cela nous permet de créer un tableau `corres` permettant de faire la correspondance entre la valeur d'un élément du tableau `t` et son renommage. Ensuite, on utilise une référence `num` permettant de faire un nouveau renommage le cas échéant, et pour chaque valeur dans le tableau `t`, si elle n'a pas encore de correspondance, on en rajoute une, puis on fait le renommage dans le tableau `t'`. Pour un tableau `t`, les deux boucles sont de complexité $\mathcal{O}(|t|)$, et la création du tableau `corres` est de complexité $\mathcal{O}(\max t)$. La complexité totale est donc $\mathcal{O}(|t| + \max t)$.

Remarque

> Notons que si on avait utilisé un dictionnaire plutôt qu'un tableau pour `corres`, sa création et ses modifications auraient été en $\mathcal{O}(1)$ (en moyenne), et on aurait eu une complexité totale en $\mathcal{O}(|t|)$ en moyenne.

28. L'idée de cette fonction est de mettre à jour les représentants au fur et à mesure qu'on parcourt les paires d'états. Initialement, on suppose que chaque état est son propre représentant, et chaque fois qu'on rencontre deux états équivalents, on indiquera que le représentant de chacun des deux devient le plus petit des deux représentants. Une fois le travail terminé, on fait appel à la fonction précédente pour renommer tous les états. On rappelle que `Array.init n f` crée le tableau `t` de taille `n` tel que `t.(i)` vaut `f i`. Pour ceux qui savent la reconnaître, il s'agit (à peu de chose près) de la structure Union-Find (ici uniquement Union) qui sert à gérer des partitions d'ensembles.

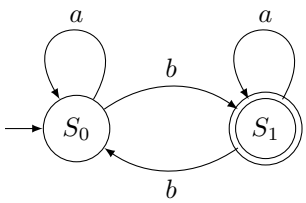
```
let relation phi psi =
    let n = Array.length phi in
    let t = Array.init n (fun i -> i) in
    let equiv p q = phi.(p) = phi.(q) || psi.(p) = psi.(q) in
    for p = 0 to n - 1 do
        for q = 0 to n - 1 do
            if equiv p q then
               let r = min t.(p) t.(q) in
               (t.(p) <- r; t.(q) <- r)
        done
    done;
    renomme t;;
```

29. D'après la question 21, il existe un automate $\mathcal{D}$ (qui est la partie accessible de $\mathcal{A} \times \mathcal{B}$) tel qu'il existe deux morphismes φ de $\mathcal{D}$ vers $\mathcal{A}$ et φ' de $\mathcal{D}$ vers $\mathcal{B}$. D'après les questions 25 et 26, il existe un automate $\mathcal{C}$ et deux morphismes ψ de $\mathcal{A}$ vers $\mathcal{C}$ et ψ' de $\mathcal{B}$ vers $\mathcal{C}$.

30. L'automate $\mathcal{D}$ est l'automate construit à la question 18. Les morphismes φ et φ' sont définis par :

q	$\varphi(q)$	$\varphi'(q)$
EH	E	H
FI	F	I
GJ	G	J
FH	F	H
GK	G	K

On en déduit que les classes d'équivalence sont $S_0 = \{EH, FH, FI\}$ et $S_1 = \{GJ, GK\}$. L'automate $\mathcal{C}$ est un automate à deux états, S_0 est l'état initial et S_1 est un état final :

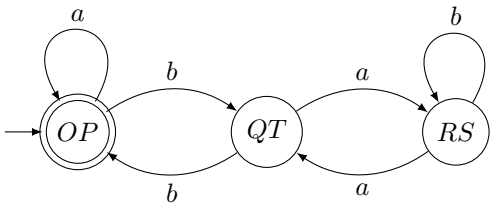


Les morphismes ψ et ψ' sont donnés par :

q	$\psi(q)$
E	S_0
F	S_0
G	S_1

q	$\psi'(q)$
H	S_0
I	S_0
J	S_1
K	S_1

31. Soient $\mathcal{A}$ et $\mathcal{A}'$ deux automates de $\mathfrak{K}_L$ ayant m_L états. Alors $\mathcal{A}$ et $\mathcal{A}'$ sont accessibles par hypothèse. Par la question 29, il existe un automate $\mathcal{C}$ et deux morphismes $\varphi : \mathcal{A} \to \mathcal{C}$ et $\psi : \mathcal{A}' \to \mathcal{C}$. Par la question 13, $\mathcal{C} \in \mathfrak{K}_L$; par surjectivité de φ, $\mathcal{C}$ a moins d'états que $\mathcal{A}$, et par minimalité de m_L, on en déduit que $\mathcal{C}$ a également m_L états. Par la question 14, $\mathcal{C}$ est isomorphe avec $\mathcal{A}$ et avec $\mathcal{C}$. Sachant que $\psi^{-1} \circ \varphi$ est un morphisme de $\mathcal{A}$ vers $\mathcal{A}'$ (question 15), on en déduit que $\mathcal{A}$ et $\mathcal{A}'$ sont isomorphes.
32. Soit $\mathcal{A} \in \mathfrak{K}_L$ et $\mathcal{A}' \in \mathfrak{K}_L$ un automate à m_L états. Par un raisonnement similaire à la question précédente, il existe un automate $\mathcal{C} \in \mathfrak{K}_L$ et des morphismes $\varphi' : \mathcal{A} \to \mathcal{C}$ et $\psi' : \mathcal{A}' \to \mathcal{C}$. De même, $\mathcal{C}$ possède m_L états (car il en a moins que $\mathcal{A}'$ et reconnait le même langage), ce qui termine la preuve.
33. On constate que pour fusionner O et P, il faut également fusionner Q et T d'une part et on peut (ou non) fusionner R et S d'autre part. Voici un automate qui convient :



Le morphisme associé est :

q	$\phi(q)$
O	OP
P	OP
Q	QT
R	RS
S	RS
T	QT

34. Si on suppose l'existence d'un tel automate et d'un tel morphisme, en notant δ et δ' les fonctions de transition dans $\mathcal{A}_6$ et $\mathcal{A}_6^{Q,R}$, on a : $\psi(\delta(Q,b)) = \psi(O)$ est final (car O est final) et $\delta'(\psi(Q),b) = \delta'(\psi(R),b) = \psi(\delta(R,b)) = \psi(S)$ est non final (car S est non final). Ainsi, on ne peut pas avoir $\psi(\delta(Q,b)) = \delta'(\psi(Q),b)$ ce qui montre que ψ n'est pas un morphisme.
35. La question 33 n'empêchait pas de fusionner R et S, donc l'automate dessiné précédemment convient.
36. Voir corrigé DM4

$\star\star\star$