L'épreuve est composée d'un unique problème, comportant 37 questions. Après un préliminaire, ce problème est divisé en 5 parties. Pour répondre à une question, un candidat pourra réutiliser le résultat d'une question antérieure, même s'il n'est pas parvenu à démontrer ce résultat. Le but du problème est d'étudier les relations qui existent entre des automates qui reconnaissent un même langage grâce à la notion de morphismes d'automates.

Préliminaires

Concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation Caml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Enfin, si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile de tester si les hypothèses sont bien vérifiées dans le code de la fonction. Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Définition mathématique d'un automate

Définition : Dans l'ensemble du sujet, le terme *automate* désigne un automate fini déterministe complet sur l'alphabet $\{a, b\}$, c'est-à-dire un quadruplet $\mathscr{A} = \langle Q, i, \delta, F\rangle$ où Q est l'ensemble des états, i est l'état initial ($i \in Q$), $\delta : Q \times \{a, b\} \to Q$ l'application de transition et $F \subseteq Q$ est l'ensemble des états finals.
On note ε le mot vide. Par extension de δ, on appelle δ^* l'application $Q \times \{a, b\}^* \to Q$ définie pour tout état q par $\delta^*(q, \varepsilon) = q$ et, si σ est une lettre de $\{a, b\}$ et w un mot de $\{a, b\}^*$, $\delta^*(q, \sigma w) = \delta^*(\delta(q, \sigma), w)$.
Un automate $\langle Q, i, \delta, F\rangle$ est représenté par un graphe orienté. Les sommets de ce graphe sont les éléments de Q. Ce graphe admet un arc $(p, q) \in Q \times Q$ étiqueté par la lettre a si et seulement si $\delta(p, a) = q$; de même, ce graphe admet un arc $(p, q) \in Q \times Q$ étiqueté par la lettre b si et seulement si $\delta(p, b) = q$. Une flèche venant de nulle part et pointant vers i indique l'état initial. Un état final est représenté par un double cercle.

Représentation d'automates en Caml

Indication Caml : Dans toutes les questions demandant d'implémenter une fonction en Caml, on identifie l'ensemble des états Q d'un automate $\langle Q, i, \delta, F\rangle$ avec l'ensemble des entiers compris entre 0 et $|Q| - 1$. On convient de plus que l'état initial i est toujours identifié à l'entier 0. Les automates seront représentés par des triplets `(n, delta, f)` où

- `n`, de type `int`, est le nombre d'états de l'automate ; les états de l'automate sont les entiers de `0` à `n-1`,
- `delta`, de type `(int * int) array`, de longueur n, est un tableau qui stocke les couples $(\delta(q, a), \delta(q, b))_{q \in Q}$,
- `f`, de type `bool array`, de longueur n, est un tableau qui représente la fonction indicatrice de l'ensemble des états finals.

Dans l'ensemble du sujet, le type `automate` est défini par l'alias suivant.

```
type automate = int * (int * int) array * bool
```

Ci dessous sont donnés quelques exemples de sont utilisation.

- `let (n, delta, f) = aut in ...`
  permet de récupérer les composantes d'une variable `aut` de type `automate`
- `let (succ_a, succ_b) = delta.(q) in ...`
  permet ensuite de récupérer le successeur par la lettre a et le successeur par la lettre b de l'état `q` (qui est de type `int`).
- `if f.(q) then ...`
  permet de tester si l'état `q` (qui est de type `int`) est final.

Indication Caml : On rappelle que la fonction `List.lenght`, de type `'a list -> int`, renvoie la longueur d'une liste. On rappelle que `Array.make n x` permet de créer un tableau de longueur n et initialisé avec la valeur x, que `Array.copy t` renvoie une copie d'un tableau `t`, que `Array.length t` renvoie la longueur d'un tableau t. On rappelle enfin que `Array.make_matrix n m x` permet de créer un tableau de tableaux de taille $n \times m$ dont toutes les cases sont initialisées par la valeur x.

1 Premiers exemples

1. Donner, sans preuve, une description courte en langue française du langage reconnu par l'automate $\mathscr{A}_1$ de la figure 1.
2. Donner, sans preuve, une description courte en langue française du langage reconnu par l'automate $\mathscr{A}_2$ de la figure 2.
3. Donner, sans preuve, une expression rationnelle qui dénote le langage reconnu par l'automate $\mathscr{A}_1$ de la figure 1.
4. Donner, sans preuve, une expression rationnelle qui dénote le langage reconnu par l'automate $\mathscr{A}_2$ de la figure 2.
5. Écrire en Caml, sans justification, la construction d'une instance du type `automate` qui corresponde à l'automate $\mathscr{A}_2$ de la figure 2.

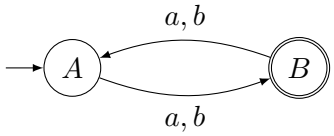


Figure 1 - Automate $\mathscr{A}_1$

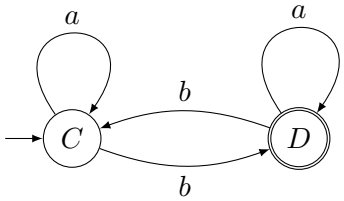


Figure 2 - Automate $\mathscr{A}_2$

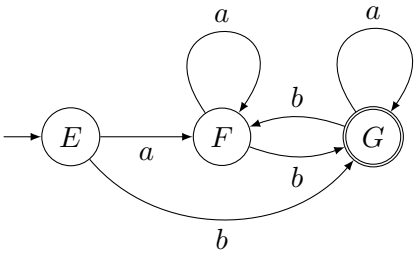


Figure 3 - Automate $\mathscr{A}_3$

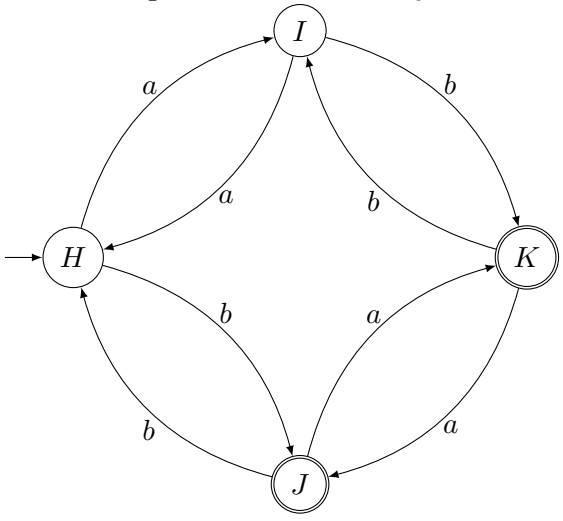


Figure 4 - Automate $\mathscr{A}_4$

2 États accessibles d'un automate

6. Écrire une fonction `numero` de type `int -> int list -> int array` qui, à partir d'un entier n et d'une liste A d'entiers compris entre 0 et $n-1$, renvoie un tableau T de taille n tel que, pour tout i compris entre 0 et $n-1$, la i-ième case $T[i]$ de T vaut -1 si i est absent de A et $T[i]$ représente l'indice de l'une des occurrences de i dans A sinon. Par exemple, `numero 5 [3;2;0];;` peut renvoyer `[|2;-1;1;0;-1|]`.

Définition : Un état q d'un automate $\mathscr{A} = \langle Q_{\mathscr{A}}, i_{\mathscr{A}}, \delta_{\mathscr{A}}, F_{\mathscr{A}} \rangle$ est dit *accessible* s'il existe un mot $w \in \{a,b\}^*$ tel que $\delta^*_{\mathscr{A}}(i_{\mathscr{A}}, w) = q$, autrement dit s'il existe un chemin qui relie l'état initial $i_{\mathscr{A}}$ à l'état q dans sa représentation graphique. (On notera que l'état initial $i_{\mathscr{A}}$ est toujours accessible.).
Soit Q' l'ensemble des états accessibles de l'automate $\mathscr{A}$. On appelle *partie accessible* de l'automate $\mathscr{A}$ le nouvel automate $\mathscr{A}' = \langle Q', i_{\mathscr{A}}, \delta', F_{\mathscr{A}} \cap Q' \rangle$ où δ' est la restriction de l'application $\delta_{\mathscr{A}}$ au domaine $Q' \times \{a,b\}$. On dit qu'un automate est *accessible* lorsque tous ses états sont accessibles.

7. Écrire une fonction `etats_accessibles`, de type `automate -> int list`, qui renvoie la liste des états accessibles de l'automate donné en argument et que l'on obtient par un parcours du graphe en profondeur depuis l'état initial. La liste renvoyée doit suivre l'ordre dans lequel les états sont rencontrés pour la première fois et ne doit pas contenir de doublons. Donner la complexité de la fonction écrite.
8. Écrire une fonction `partie_accessibles` de type `automate -> automate` qui construit la partie accessible de l'automate donné en argument. On pourra réemployer les fonctions implémentées aux questions 6 et 7.

3 Morphismes d'automates

Définition : Soient deux automates $\mathscr{A} = \langle Q_{\mathscr{A}}, i_{\mathscr{A}}, \delta_{\mathscr{A}}, F_{\mathscr{A}} \rangle$ et $\mathscr{B} = \langle Q_{\mathscr{B}}, i_{\mathscr{B}}, \delta_{\mathscr{B}}, F_{\mathscr{B}} \rangle$. Une application $\varphi : Q_{\mathscr{A}} \to Q_{\mathscr{B}}$ est appelée *morphisme d'automates* de l'automate $\mathscr{A}$ vers l'automate $\mathscr{B}$, et est noté $\varphi : \mathscr{A} \to \mathscr{B}$, si elle satisfait les conditions suivantes :

$$\varphi \text{ est surjective,} \tag{1}$$

$$\varphi(i_{\mathscr{A}}) = i_{\mathscr{B}} \tag{2}$$

$$\forall q \in Q_{\mathscr{A}}, \quad \forall \sigma \in \{a,b\}, \quad \varphi(\delta_A(q,\sigma)) = \delta_{\mathscr{B}}(\varphi(q),\sigma) \tag{3}$$

$$\forall q \in Q_{\mathscr{A}}, \quad q \in F_{\mathscr{A}} \iff \varphi(q) \in F_{\mathscr{B}} \tag{4}$$

Indication Caml : En Caml, on représente un morphisme $\varphi : \mathscr{A} \to \mathscr{B}$ par le tableau $[\varphi(q)]_{q \in Q_{\mathscr{A}}}$ de longueur $|Q_{\mathscr{A}}|$, de type `int array`, formé d'entiers compris entre 0 et $|Q_{\mathscr{B}}| - 1$. On pourra utiliser le type `morphisme`, défini par l'alias

```
type morphisme = int array;;
```

3.1 Exemples de morphismes d'automates

9. À partir des figures 2 et 3 représentant les automates $\mathscr{A}_2$ et $\mathscr{A}_3$, recopier le tableau suivant et le compléter sans justification par des états de $\mathscr{A}_2$ de sorte que ce tableau représente un morphisme d'automates φ de l'automate $\mathscr{A}_3$ vers l'automate $\mathscr{A}_2$.

q	$\varphi(q)$
E	
F	
G	

10. À partir des figures 2 et 4 représentant les automates $\mathscr{A}_2$ et $\mathscr{A}_4$, donner, sans en justifier l'expression, un morphisme d'automates de l'automate $\mathscr{A}_4$ vers l'automate $\mathscr{A}_2$.
11. À partir des figures 1 et 2, montrer qu'il n'existe pas de morphisme d'automates de l'automate $\mathscr{A}_1$ vers l'automate $\mathscr{A}_2$.

12. À partir des figures 2 et 5, montrer qu'il n'existe pas de morphisme d'automates de l'automate $\mathscr{A}_5$ vers l'automate $\mathscr{A}_2$.

3.2 Propriétés des morphismes d'automates

13. Montrer que deux automates acceptent le même langage dès lors qu'il existe un morphisme d'automates de l'un des automates vers l'autre.
14. Montrer qu'un morphisme φ entre deux automates ayant le même nombre d'états est nécessairement une application bijective et que l'application φ^{-1} est encore un morphisme d'automates.

On dit dans ce cas que φ est un **isomorphisme** d'automates.

15. Montrer que la composition de deux morphismes d'automates est encore un morphisme d'automates.

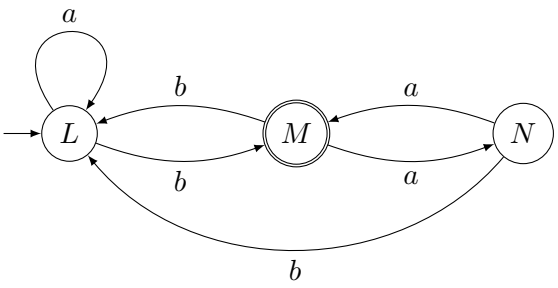


Figure 5 - Automate $\mathscr{A}_5$

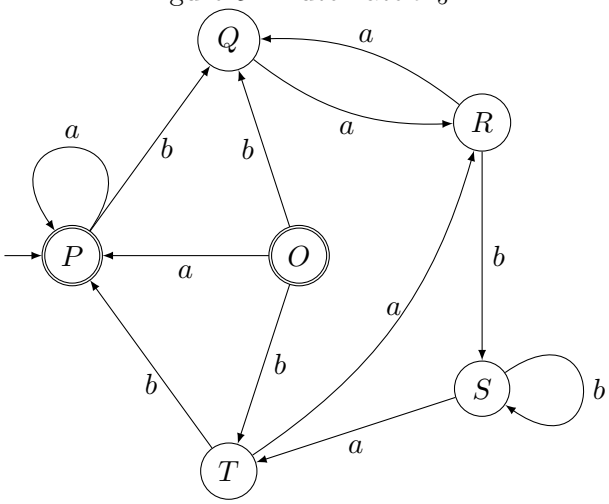


Figure 6 - Automate $\mathscr{A}_6$

3.2 Existence de morphismes d'automates entre automates accessibles

16. Montrer que le point (1) de la définition des morphismes d'automates découle des points (2), (3) et (4) quand les deux automates considérés sont accessibles.
17. Écrire en Caml une fonction `existe_morphisme` de type `automate -> automate -> bool*morphisme` qui, sous l'hypothèse que les deux automates en argument sont accessibles, renvoie d'une part un booléen indiquant l'existence d'un morphisme d'automates du premier argument vers le second et d'autre part un tel morphisme lorsqu'il existe. Lorsqu'un tel morphisme n'existe pas, la seconde composante de la valeur de retour est un tableau quelconque. On pourra expliquer le principe de l'algorithme avant d'en donner le code.

4 Constructions de morphismes d'automates

4.1 Automate produit

Définition : Soient $\mathscr{A} = \langle Q_{\mathscr{A}}, i_{\mathscr{A}}, \delta_{\mathscr{A}}, F_{\mathscr{A}} \rangle$ et $\mathscr{A}' = \langle Q_{\mathscr{A}'}, i_{\mathscr{A}'}, \delta_{\mathscr{A}'}, F_{\mathscr{A}'} \rangle$ deux automates. On appelle *automate produit*, et on note $\mathscr{A} \times \mathscr{A}'$, le nouvel automate

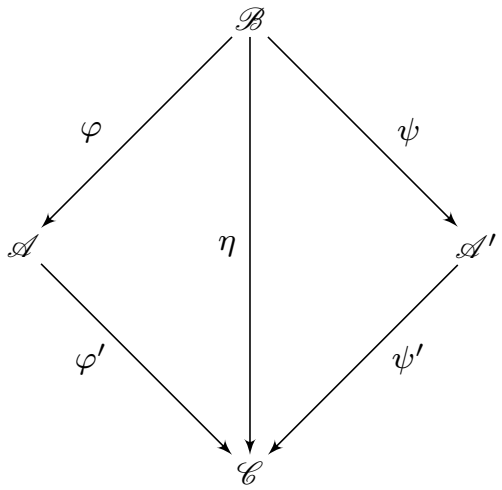
$$\mathscr{A} \times \mathscr{A}' = \langle Q_{\mathscr{A}} \times Q_{\mathscr{A}'}, (i_{\mathscr{A}}, i_{\mathscr{A}'}), \delta_{\mathscr{A}\times\mathscr{A}'}, F_{\mathscr{A}} \times F_{\mathscr{A}'} \rangle$$

où l'application $\delta_{\mathscr{A}\times\mathscr{A}'}$ est définie, pour tout couple d'états $(q, q') \in Q_{\mathscr{A}} \times Q_{\mathscr{A}'}$ et pour toute lettre $\sigma \in \{a, b\}$, par $\delta_{\mathscr{A}\times\mathscr{A}'}((q, q'), \sigma) = (\delta_{\mathscr{A}}(q, \sigma), \delta_{\mathscr{A}'}(q', \sigma))$.

18. On considère les automates $\mathscr{A}_3$ et $\mathscr{A}_4$ qui sont représentés par les figures 3 et 4. Dessiner, sans justification, la partie accessible du produit d'automates $\mathscr{A}_3 \times \mathscr{A}_4$.
19. Écrire une fonction `produit` de type `automate -> automate -> automate` qui renvoie le produit des deux automates donnés en argument.
20. Soit (q, q') un état accessible du produit de deux automates qui acceptent le même langage. Montrer que q est un état final du premier automate si et seulement si q' est un état final du second automate.
21. Montrer qu'il existe toujours un morphisme d'automates de la partie accessible du produit de deux automates accessibles qui acceptent le même langage vers chacun de ces deux automates.

4.2 Diagramme d'automates

Dans toute la sous-section 4.2, on considère qu'il existe trois automates accessibles $\mathscr{A}$, $\mathscr{A}'$ et $\mathscr{B} = \langle Q_{\mathscr{B}}, i_{\mathscr{B}}, \delta_{\mathscr{B}}, F_{\mathscr{B}} \rangle$ et deux morphismes d'automates $\varphi : \mathscr{B} \to \mathscr{A}$ et $\psi : \mathscr{B} \to \mathscr{A}'$. Le but de cette sous-section est de construire un nouvel automate accessible $\mathscr{C}$ et trois morphismes φ', ψ' et η dont la situation est résumée dans le diagramme suivant.



Définition : On définit une relation sur $Q_{\mathscr{B}}$, notée $\equiv$. Pour tout couple d'états (p, q) appartenant à $Q^2_{\mathscr{B}}$, $p \equiv q$ s'il existe une suite finie de longueur $k + 1$ (avec $k \in \mathbb{N}$) constituée des termes $p = q_0, q_1, q_2, ..., q_k = q$ d'états de $Q_{\mathscr{B}}$ telle que

$$\forall 0 \leq j < k, \quad \varphi(q_j) = \varphi(q_{j+1}) \text{ ou } \psi(q_j) = \psi(q_{j+1}).$$

22. Montrer que la relation $\equiv$ définie sur l'ensemble $Q_{\mathscr{B}}$ est une relation d'équivalence.
23. Montrer que, pour tout couple d'états $(p, q) \in Q^2_{\mathscr{B}}$, si $p \equiv q$, alors, pour toute lettre $\sigma \in \{a, b\}$, on a $\delta_{\mathscr{B}}(p, \sigma) \equiv \delta_{\mathscr{B}}(q, \sigma)$.
24. Montrer que, pour tout couple d'états $(p, q) \in Q^2_{\mathscr{B}}$, si $p \equiv q$, alors p est un état final de l'automate $\mathscr{B}$ si et seulement si q est un état final de l'automate $\mathscr{B}$.

Définition : La classe d'équivalence, notée $[q]$, d'un état $q \in Q_{\mathscr{B}}$ est l'ensemble

$$[q] = \{p \in Q_{\mathscr{B}}; q \equiv p\}.$$

Dans ce qui suit, on appelle ℓ le nombre de classes d'équivalence de la relation $\equiv$ et on note $S_0, S_1, ..., S_{\ell-1}$ ces classes. On choisira S_0 de sorte que $i_{\mathscr{B}} \in S_0$. On note η l'application $Q_{\mathscr{B}} \to \{S_0, S_1, ..., S_{\ell-1}\}$ qui, à chaque état $q \in Q_{\mathscr{B}}$, associe la classe d'équivalence $[q]$.

Indication Caml : En Caml, on représentera la classe d'équivalence S_j par l'indice j.

25. Construire un automate accessible $\mathscr{C}$ dont l'ensemble d'états est $\{S_0, S_1, ..., S_{\ell-1}\}$ et tel que η est un morphisme d'automates de l'automate $\mathscr{B}$ vers l'automate $\mathscr{C}$. Justifier.
26. Construire deux morphismes d'automates $\varphi' : \mathscr{A} \to \mathscr{C}$ et $\psi' : \mathscr{A}' \to \mathscr{C}$, où $\mathscr{C}$ est l'automate construit à la question 25.
27. Écrire une fonction `renomme`, de type `int array -> int array`, qui renomme le contenu d'un tableau contenant des entiers positifs prenant ℓ valeurs distinctes en utilisant les entiers entre 0 et $\ell - 1$. Le premier élément du résultat doit de plus être égal à 0. Par exemple `renomme` $[|4;4;5;0;4;5|]$`;;` peut renvoyer $[|0;0;1;2;0;1|]$. Préciser la complexité de la fonction proposée.
28. Écrire une fonction `relation`, de type `morphisme -> morphisme -> morphisme`, qui, à partir des tableaux $[\varphi(q)]_{q\in Q_{\mathscr{B}}}$ et $[\psi(q)]_{q\in Q_{\mathscr{B}}}$, renvoie le tableau $[\eta(q)]_{q\in Q_{\mathscr{B}}}$, autrement dit, qui renvoie un tableau `t` d'entiers compris entre 0 et $\ell - 1$ tel que pour tout couple d'états $(p, q) \in Q^2_{\mathscr{B}}$, les valeurs `t.(q)` et `t.(p)` sont égales si et seulement si $p \equiv q$ et tel que `t.(0)` vaut 0.

5 Réduction d'automates

5.1 Existence et unicité

29. Montrer que si deux automates accessibles $\mathscr{A}$ et $\mathscr{A}'$ acceptent le même langage, alors on peut construire un automate $\mathscr{C}$ et deux morphismes $\varphi' : \mathscr{A} \to \mathscr{C}$ et $\psi' : \mathscr{A}' \to \mathscr{C}$.
30. Déterminer l'automate $\mathscr{C}$ défini à la question 29 pour les automates $\mathscr{A}_3$ et $\mathscr{A}_4$ (figures 3 et 4) et préciser les applications φ' et ψ'.

Soit L un langage rationnel et $\mathcal{R}_L$ l'ensemble des automates (complets déterministes) accessibles qui acceptent le langage L. On note m_L le plus petit nombre d'états d'un automate de $\mathcal{R}_L$.

31. Montrer que deux automates de $\mathcal{R}_L$ ayant m_L états sont nécessairement isomorphes.
32. Montrer que, pour tout automate $\mathscr{A}$ dans $\mathcal{R}_L$, il existe un morphisme $\varphi : \mathscr{A} \to \mathcal{M}_L$ où $\mathcal{M}_L$ est un automate de $\mathcal{R}_L$ à m_L états.

5.2 Construction d'un automate réduit par fusion d'états

Définition : Soient $\mathscr{A} = \langle Q_{\mathscr{A}}, i_{\mathscr{A}}, \delta_{\mathscr{A}}, F_{\mathscr{A}}\rangle$ et $\mathscr{A}' = \langle Q_{\mathscr{A}'}, i_{\mathscr{A}'}, \delta_{\mathscr{A}'}, F_{\mathscr{A}'}\rangle$ deux automates. On dit que deux états p et q de l'automate $\mathscr{A}$ ont été fusionnés dans l'automate $\mathscr{A}'$ s'il existe un morphisme d'automates de $\mathscr{A}$ vers $\mathscr{A}'$ tel que $\varphi(p) = \varphi(q)$ et si le nombre d'états satisfait $|Q_{\mathscr{A}'}| < |Q_{\mathscr{A}}|$.

33. On considère l'automate $\mathscr{A}_6$ de la figure 6. Dessiner un automate $\mathscr{A}_6^{O,P}$ dans lequel les états O et P ont été fusionnés. On donnera un morphisme d'automates $\mathscr{A}_6 \to \mathscr{A}_6^{O,P}$.
34. Expliquer brièvement pourquoi il n'est pas possible de construire un automate $\mathscr{A}_6^{Q,R}$ muni d'un morphisme d'automates $\psi : \mathscr{A}_6 \to \mathscr{A}_6^{Q,R}$ tel que $\psi(Q) = \psi(R)$.
35. Quels états faut-il encore fusionner dans $\mathscr{A}_6^{O,P}$ pour obtenir un automate à trois états $\mathcal{M}_{L_6}$, qui reconnaît le même langage que $\mathscr{A}_6$?
36. Soit $\mathscr{A} = \langle Q, i, \delta, F\rangle$ un automate accessible. On appelle P le graphe orienté de sommets $Q\times Q$ et, pour toute lettre $\sigma \in \{a, b\}$ et tout sommet $(p, q) \in Q\times Q$, d'arcs allant du sommet $(\delta(p, \sigma), \delta(q, \delta)) \in Q\times Q$ vers le sommet (p, q).

    Écrire en Caml une fonction `table_de_predecesseur` de type `automate -> bool array array` qui prend en entrée un automate accessible $\mathscr{A} = \langle Q, i, \delta, F\rangle$ à n états et renvoie en sortie une matrice de taille $n \times n$. Pour tous les états p et q de l'automate $\mathscr{A}$, la valeur de la case (p, q) de la matrice vaut `true` si et seulement s'il existe deux états $(p_0, q_0) \in Q^2$ et un chemin du sommet (p_0, q_0) vers le sommet (p, q) dans le graphe P tel que $p_0 \in F$ et $q_0 \notin F$ ou $p_0 \notin F$ et $q_0 \in F$.

    On essaiera de ne pas dépasser une complexité en $O(n^2)$.
37. En décrire le principe, le justifier, puis écrire en Caml une fonction `reduit`, de type `automate -> automate` qui prend en entrée un automate $\mathscr{A}$ et renvoie l'automate $\mathcal{M}_L$ associé au langage L reconnu par $\mathscr{A}$.