DM2 - Mines MP 2019

1 Premiers exemples

1. Le langage reconnu par l'automate A_1 est l'ensemble des $\boxed{\text{mots de taille impaire}}$.
2. Le langage reconnu par l'automate A_2 est l'ensembles des $\boxed{\text{mots contenant un nombre impair de } b}$.
3. $\boxed{L(A_1) = (a \cdot a|b \cdot b|a \cdot b|b \cdot a)^* \cdot (a|b)}$
4. $\boxed{L(A_1) = (a|b \cdot a^* \cdot b)^* \cdot b \cdot a^*}$
5.
```
let a2 = 2, [|(0, 1) ; (1, 0)|], [|false; true|]  ;;
```

2 États accessibles d'un automate

6.
```
let numero n a =
    let t = Array.make n (-1) in
    let rec aux index = function (*parcourt la liste*)
      |[] -> ()
      |h::q -> t.(h) <- index ; aux (index + 1) q in
    aux 0 a ; t ;;
```

7.
```
let etats_accessibles aut =
    let n, delta, f = aut in
    let visited = Array.make n false in
    let parcours = ref [] in
    let aux etat =
        if not visited.(etat) then
            begin
            visited.(etat) <- true ;
            parcours := etat :: !parcours ;
            let succ_a, succ_b = delta.(etat) in
            aux succ_a ;
            aux succ_b
            end
    in
    aux 0 ;;
    List.rev !parcours
```

Compléxité : La création d'une `Array` de taille n est en $O(n)$, et l'accès à un élément d'une liste, tout comme la concatenation d'un élément avec une liste, sont en $O(1)$. On ne s'intéresse donc qu'aux appels récursifs de la fonction `aux`. Comme $|Q| = n$, et que pour tout $q \in Q$, il existe deux états q_1 et q_2 (potentiellement égaux) tels que $\delta(q, a) = q_1$ et $\delta(q, b) = q_2$, la fonction `aux` fait au plus $2n$ appels récursifs, soit une compléxité en $\boxed{O(n)}$.

8.
```
let partie_accessible aut =
    let n, delta, f = aut in
    let new_etats = etats_accessibles aut in
    let apparition = numero n new_etats in
    let new_n = List.length new_etats in
    let new_delta = Array.make new_n (0,0) in
    let new_f = Array.make new_n false in
    let rec aux = function
        |[] -> (new_n, new_delta, new_f)
        |h::t -> let s = apparition.(h) in
                new_f.(s) <- f.(h);
                let succ_a, succ_b = delta.(h) in
                new_delta.(s) <- apparition.(succ_a) ,apparition.(succ_b);
                aux t
    in
    aux new_etats ;;
```

3 Morphismes d'automates

3.1 Exemples de morphismes d'automates

9. $\varphi : \mathcal{A}_3 \to \mathcal{A}_2$ est représentée par

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

10. $\varphi : \mathcal{A}_4 \to \mathcal{A}_2$ est représentée par

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

11. Supposons qu'il existe un morphisme d'automates $\varphi : \mathcal{A}_1 \to \mathcal{A}_2$, alors, pour vérifier (2) : $\varphi(A) = C$, et pour vérifier (4) : $\varphi(B) = D$. Or si φ est un morphisme d'automates, alors d'après (3),

$$\begin{aligned} \varphi(\delta_{\mathcal{A}_1}(A, a)) &= \varphi(B) \\ &= D \\ &= \delta_{\mathcal{A}_2}(\varphi(A), a) \end{aligned}$$

Mais, $\varphi(A) = C$ d'après (2), donc

$$\begin{aligned} \delta_{\mathcal{A}_2}(\varphi(A), a) &= \delta_{\mathcal{A}_2}(C, a) \\ &= C \\ &= D \\ &\to \underline{\text{Absurde !}} \end{aligned}$$

Ainsi, $\boxed{\text{il n'existe pas de morphisme d'automates de } \mathcal{A}_1 \text{ vers } \mathcal{A}_2}$.

12. Comme à la question précédente, supposons qu'il existe un morphisme d'automates $\varphi : \mathcal{A}_5 \to \mathcal{A}_2$, alors (2) et (4) nous donne : $\varphi(L) = C$, $\varphi(M) = D$ et $\varphi(N) = C$ (car $N \notin F_{\mathcal{A}_5}$). (3) impose alors,

$$\begin{aligned} \varphi(\delta_{\mathcal{A}_5}(N, b)) &= \varphi(L) \\ &= C \\ &= \delta_{\mathcal{A}_2}(\varphi(N), b) \\ &= \delta_{\mathcal{A}_2}(C, b) \quad (\varphi(N) = C) \\ &= D \\ &\to \underline{\text{Absurde !}} \end{aligned}$$

Ainsi, $\boxed{\text{il n'existe pas de morphisme d'automates de } \mathcal{A}_5 \text{ vers } \mathcal{A}_2}$.

3.2 Propriétés des morphismes d'automates

13. Soit $\mathcal{A}, \mathcal{B}$ deux automates. Supposons qu'il existe un morphisme d'automates $\varphi : \mathcal{A} \to \mathcal{B}$.
Notons $\mathcal{P}(n)$ le prédicat : « pour tout $q \in Q_{\mathcal{A}}$ et pour tout mot m de taille n, $\varphi(\delta^*_{\mathcal{A}}(q, m)) = \delta^*_{\mathcal{B}}(\varphi(q), m)$ ».
Montrons par récurrence simple, que pour tout $n \in \mathbb{N}$, $\mathcal{P}(n)$.

<u>Initialisation</u> : Si $n = 0$, alors $m(= \varepsilon)$ est le mot vide, alors pour tout état q,
$\delta^*_{\mathcal{A}}(q, \varepsilon) = q$ et $\delta^*_{\mathcal{B}}(\varphi(q), \varepsilon) = \varphi(q)$ donc $\varphi(\delta^*_{\mathcal{A}}(q, \varepsilon)) = \varphi(q) = \delta^*_{\mathcal{B}}(\varphi(q), \varepsilon)$.

<u>Hérédité</u> : Soit $n \in \mathbb{N}$, supposons $\mathcal{P}(n)$, montrons $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$,
Soit $q \in Q_{\mathcal{A}}$ un état quelconque et m un mot de taille $n+1$, on peut alors noter $m = \sigma m_r$ où m_r est un mot de taille n, et σ une lettre dans $\{a, b\}$. Ainsi,

$$\begin{aligned} \varphi(\delta^*_{\mathcal{A}}(q, m)) &= \varphi(\delta^*_{\mathcal{A}}(\delta_{\mathcal{A}}(q, \sigma), m_r)) \text{ par définition de } \delta^*_{\mathcal{A}} \\ &= \delta^*_{\mathcal{B}}(\varphi(\delta_{\mathcal{A}}(q, \sigma)), m_r) \text{ par l'hypothèse de récurrence} \\ &= \delta^*_{\mathcal{B}}(\delta_{\mathcal{B}}(\varphi(q), \sigma), m_r) \text{ par la propriété (3) d'un morphisme d'automates} \\ &= \delta^*_{\mathcal{B}}(\varphi(q), m) \text{ par définition de } \delta^*_{\mathcal{B}} \end{aligned}$$

Donc $\forall n \in \mathbb{N}$, $\mathcal{P}(n)$ est vérifié.
Ainsi, si $m \in L(\mathcal{A})$, alors $\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, m) \in F_{\mathcal{A}}$, et donc $\varphi(\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, m)) = \delta^*_{\mathcal{B}}(\varphi(i_{\mathcal{A}}), m) = \delta^*_{\mathcal{B}}(i_{\mathcal{B}}, m) \in F_{\mathcal{B}}$

d'après ce que l'on vient de montrer, (2) et (4). Donc $\delta^*_{\mathcal{B}}(i_{\mathcal{B}}, m) \in F_{\mathcal{B}}$, et par conséquent $m \in L(\mathcal{B})$. $\boxed{\text{D'où le résultat}}$.

14. Soit $\mathcal{A}, \mathcal{B}$ deux automates **finis**, tels que $|Q_{\mathcal{A}}| = |Q_{\mathcal{B}}|$. Supposons qu'il existe un morphisme d'automates $\varphi : \mathcal{A} \to \mathcal{B}$, alors φ est surjective par définition. Or comme $|Q_{\mathcal{A}}| = |Q_{\mathcal{B}}|$, φ surjective $\Leftrightarrow$ φ bijective. Donc $\boxed{\varphi \text{ est nécessairement bijective}}$.
De plus :
    (1) φ^{-1} est bijective donc surjective ;
    (2) $\varphi^{-1}(i_{\mathcal{B}}) = \varphi^{-1}(\varphi(i_{\mathcal{B}})) = i_{\mathcal{A}}$;
    (3) Soit $q \in F_{\mathcal{B}}$, d'après la définition de $\varphi, \forall \sigma \in \{a, b\}$, $\varphi(\delta_{\mathcal{A}}(\varphi^{-1}(q), \sigma)) = \delta_{\mathcal{B}}(q, \sigma)$, et donc en appliquant φ^{-1} à l'égalité :
    $$\forall q \in Q_{\mathcal{B}},\ \forall \sigma \in \{a, b\},\ \varphi^{-1}(\delta_{\mathcal{B}}(q, \sigma)) = \delta_{\mathcal{A}}(\varphi^{-1}(q), \sigma)$$
    (4) $\forall q \in Q_{\mathcal{B}}$, (encore une fois d'après la définition de φ),
    $$\begin{aligned} \varphi^{-1}(q) \in F_{\mathcal{A}} &\iff \varphi(\varphi^{-1}(q)) \in F_{\mathcal{B}} \\ &\iff q \in F_{\mathcal{B}} \end{aligned}$$
Donc $\boxed{\varphi^{-1} \text{ est bien un morphisme d'automates}}$.

15. Soit $\mathcal{A}, \mathcal{B}, \mathcal{C}$ trois automates et $\varphi : \mathcal{B} \to \mathcal{C}$, $\psi : \mathcal{A} \to \mathcal{B}$ deux morphismes d'automates.
    (1) φ et ψ surjectives donc $\varphi \circ \psi$ est surjective ;
    (2) $(\varphi \circ \psi)(i_{\mathcal{A}}) = \varphi(i_{\mathcal{B}}) = i_{\mathcal{C}}$;
    (3) $\forall q \in Q_{\mathcal{A}}, \forall \sigma \in \{a, b\}$, $(\varphi \circ \psi)(\delta_{\mathcal{A}}(q, \sigma)) = \varphi(\delta_{\mathcal{B}}(\psi(q), \sigma)) = \delta_{\mathcal{A}}((\varphi \circ \psi)(q), \sigma)$;
    (4) $\forall q \in Q_{\mathcal{A}}$, $q \in F_{\mathcal{A}} \iff \psi(q) \in F_{\mathcal{B}} \iff (\varphi \circ \psi)(q) \in F_{\mathcal{C}}$;
Donc $\varphi \circ \psi : \mathcal{A} \to \mathcal{C}$ est un morphisme d'automates, $\boxed{\text{d'où le résultat}}$.

3.3 Existence de morphismes d'automates entre automates accessibles

16. Soit $\mathcal{A}, \mathcal{B}$ deux automates **accessibles**, soit $\varphi : \mathcal{A} \to \mathcal{B}$ une application verifiant les propriétés (2), (3) et (4). Le résulat montré par récurrence à la question 13. est toujours valable puisque les seules hypothèses utilisées ((2), (3) et (4)) sont vérfiées.
Soit $q \in Q_{\mathcal{B}}$, comme q est accesible, il existe $m \in L(\mathcal{B})$ tel que :
$$\begin{aligned} \delta^*_{\mathcal{B}}(i_{\mathcal{B}}, m) &= q \\ &= \delta^*_{\mathcal{B}}(\varphi(i_{\mathcal{A}}), m) \text{ d'après } (2) \\ &= \varphi(\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, m)) \text{ d'après la question 13.} \end{aligned}$$
Or, comme $\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, m) \in Q_{\mathcal{A}}$, alors il existe $q' \in Q_{\mathcal{A}}$ tel que $\varphi(q') = q$ donc φ est surjective, d'où $\boxed{(\text{automates accessibles}) \wedge (2) \wedge (3) \wedge (4) \implies (1)}$

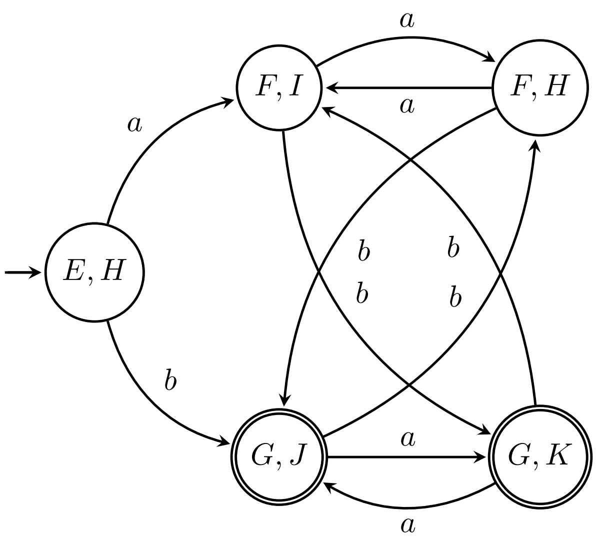
17.
```
let existe_morphismes aut1 aut2 =
    let n1, delta1, f1 = aut1 in
    let n2, delta2, f2 = aut2 in
    let def = ref true in (*variable indiquant si un morphisme existe ou non*)
    let visited = Array.make n1 false in
    let morphisme = Array.make n1 (-1) in
    let construire etat1 etat2 = (*fonction qui construit le morphisme*)
        if morphisme.(etat1) <> 1 (*si phi(q) ne s'est pas encore vu etre associe
        ↪ une image, on la definit*)
            then begin
                if ( f1.(etat1) = f2.(etat2) ) || ( (not f1.(etat1)) = (not
                ↪ f2.(etat2)) ) (*test de la condition (4)*)
                then morphisme.(etat1) <- etat2 (*si (4) est respectee, alors on
                ↪ peut definir phi(q)*)
                else def := false (*sinon il n'existe pas de morphisme*)
            end
        else if morphisme.(etat1) <> etat2 (*si phi(q) est deja defini, mais que la
        ↪ condition (3) n'est pas respectee...*)
             then def := false (*... alors il n'existe pas de morphisme*)
```

```
    in
    let rec aux etat1 = (*fonction qui parcours l'automate afin de constuire le morphisme*)
        if not visited.(etat1) (*on verifie que le sommet n'a pas deja ete visite*)
        then begin
            let etat2 = morphisme.(etat1) in
            let succ1_a, succ1_b = delta1.(etat1) in (*on construit le morphisme recursivement, en partant du sommet que l'on visite*)
            let succ2_a, succ2_b = delta2.(etat2) in
            visited.(etat1) <- true ;
            construire succ1_a succ2_a ; (*on definit les images de phi pour sigma = a*)
            construire succ1_b succ2_b ; (*on definit les images de phi pour sigma = b*)
            if !def then (aux succ1_a ; aux succ1_b) (*si le morphisme existe (i.e les etapes de construction se sont achevees), alors on continue jusqu'a ce que tous les sommets soient visites*)
            end
    in
    morphisme.(0) <- 0 ; (*initialisation de la constuction avec la condition (2)*)
    aux 0 ; (*début du parcours*)
    !def , morphisme ;;
```

4 Construction de morphismes d'automates

4.1 Automate produit

18.



Partie accessible de $\mathcal{A}_3 \times \mathcal{A}_4$

19.

```
let produit aut1 aut2 =
    let n1, delta1, f1 = aut1 in
    let n2, delta2, f2 = aut2 in
    let n = n1 * n2 in
    let f = Array.make n false in
    let delta = Array.make n (0, 0) in
    let prod_etats n x1 x2 = (*fonction auxiliaire pour calculer delta d'un produit d'automates*)
    match x1, x2 with
    |(a, b), (c, d) -> n*a + c, n*b + d in
    for i = 0 to n1 - 1 do
        for j = 0 to n2 - 1 do
            begin
            if f1.(i) && f2.(j) then
```

```
                f.(n2 * i + j) <- true ;
                delta.(n2 * i + j) <- prod_etats n2 delta1.(i) delta2.(j) ;
                end
            done; done;
        partie_accessible (n, delta, f) ;;
```

20. Soit $\mathcal{A}, \mathcal{A}'$ deux automates.
Notons $\mathcal{P}(n)$ le prédicat : « pour tout état $(q, q') \in Q_{\mathcal{A}} \times Q_{\mathcal{A}'}$ et pour tout mot m de taille n, $\delta^*_{\mathcal{A}\times\mathcal{A}'}((q, q'), m) = (\delta^*_{\mathcal{A}}(q, m), \delta^*_{\mathcal{A}'}(q', m))$ ». Montrons par récurrence simple, que pour tout $n \in \mathbb{N}$, $\mathcal{P}(n)$.

Initialisation : Si $n = 0$, alors $m(= \varepsilon)$ est le mot vide, alors pour tout état (q, q'), $\delta^*_{\mathcal{A}\times\mathcal{A}'}((q, q'), \varepsilon) = (q, q')$
Or $\delta^*_{\mathcal{A}}(q, \varepsilon) = q$ et $\delta^*_{\mathcal{A}'}(q', \varepsilon) = q'$ donc $\delta^*_{\mathcal{A}\times\mathcal{A}'}((q, q'), \varepsilon) = (q, q') = (\delta^*_{\mathcal{A}}(q, \varepsilon), \delta^*_{\mathcal{A}'}(q', \varepsilon))$

Hérédité : Soit $n \in \mathbb{N}$, supposons $\mathcal{P}(n)$, montrons $\mathcal{P}(n) \Rightarrow \mathcal{P}(n + 1)$,
Soit $(q, q') \in Q_{\mathcal{A}} \times Q_{\mathcal{A}'}$ un état quelconque et m un mot de taille $n + 1$, on peut alors noter $m = \sigma m_r$ où m_r est un mot de taille n. Ainsi,

$$\begin{aligned}
\delta^*_{\mathcal{A}\times\mathcal{A}'}((q, q'), m) &= \delta^*_{\mathcal{A}\times\mathcal{A}'}(\delta_{\mathcal{A}\times\mathcal{A}'}((q, q'), \sigma), m_r) \text{ par définition de } \delta^*_{\mathcal{A}\times\mathcal{A}'} \\
&= \delta^*_{\mathcal{A}\times\mathcal{A}'}((\delta_{\mathcal{A}}(q, \sigma), \delta_{\mathcal{A}'}(q', \sigma)), m_r) \text{ par par définition de } \delta_{\mathcal{A}\times\mathcal{A}'} \\
&= (\delta^*_{\mathcal{A}}(\delta_{\mathcal{A}}(q, \sigma), m_r), \delta^*_{\mathcal{A}'}(\delta_{\mathcal{A}'}(q', \sigma), m_r)) \text{ d'après l'hypothèse de récurrence} \\
&= (\delta^*_{\mathcal{A}}(q, m), \delta^*_{\mathcal{A}'}(q', m)) \text{ par définition de } \delta^*_{\mathcal{A}} \text{ et } \delta^*_{\mathcal{A}'}
\end{aligned}$$

Donc $\forall n \in \mathbb{N}$, $\mathcal{P}(n)$ est vérifié.
Ainsi, si $(q, q') \in Q_{\mathcal{A}} \times Q_{\mathcal{A}'}$ est un état accessible, alors il existe un mot m tel que $(q, q') = \delta^*_{\mathcal{A}\times\mathcal{A}'}((i_{\mathcal{A}}, i_{\mathcal{A}'}), m) = (\delta^*_{\mathcal{A}}(i_{\mathcal{A}}, m), \delta^*_{\mathcal{A}'}(i_{\mathcal{A}'}, m))$.
Or comme $L(\mathcal{A}) = L(\mathcal{A}')$, $m \in L(\mathcal{A}) \iff m \in L(\mathcal{A}')$. $\boxed{\text{D'où } q \in F_{\mathcal{A}} \iff q' \in F_{\mathcal{A}'}}$.

21. Soit $\mathcal{A}, \mathcal{A}'$ deux automates accessibles qui acceptent le même langage. On considère $\mathcal{B}$ la partie accessible de $\mathcal{A} \times \mathcal{A}'$.
Montrons qu'il existe un morphisme d'automates $\varphi : \mathcal{B} -> \mathcal{A}$:
Considérons $\varphi : \begin{array}{ccc} \mathcal{B} & \to & \mathcal{A} \\ (q, q') & \mapsto & q \end{array}$.
D'après la question 16, comme $\mathcal{B}$ et $\mathcal{A}$ sont deux automates accessibles, il suffit de montrer que φ vérifie (2), (3) et (4) :
   (2) Comme $i_{\mathcal{B}} = (i_{\mathcal{A}}, i_{\mathcal{A}'})$, on a directement $\varphi(i_{\mathcal{B}}) = i_{\mathcal{A}}$;
   (3) Si $(q, q') \in Q_{\mathcal{B}}$, alors pour tout $\sigma \in \{a, b\}$, on a à la fois $\varphi(q, q') = q$ par définition de φ et $\varphi(\delta_{\mathcal{B}}((q, q'), \sigma)) = \varphi(\delta_{\mathcal{A}}(q, \sigma), \delta_{\mathcal{A}'}(q', \sigma)) = \delta_{\mathcal{A}}(q, \sigma)$;
   (4) Enfin, soit $(q, q') \in Q_{\mathcal{B}}$:
      ($\Rightarrow$) Si $(q, q') \in F_{\mathcal{B}}$, par définition de $\mathcal{B}$, $\varphi(q, q') = q \in F_{\mathcal{A}}$;
      ($\Leftarrow$) Si $\varphi(q, q') = q \in F_{\mathcal{A}}$, alors d'après la question 20, $q' \in F_{\mathcal{A}'}$, et donc $(q, q') \in F_{\mathcal{B}}$;

Ainsi, on démontre bien $\boxed{\text{l'existence d'un morphisme d'automates entre } \mathcal{B} \text{ et } \mathcal{A}}$.
$\boxed{\text{La symétrie du problème entre } \mathcal{A} \text{ et } \mathcal{A}'}$ permet de conclure.

4.2 Diagramme d'automates

22. Montrons que $\equiv$ définit bien une relation d'équivalence :
   — Si $p \in Q_{\mathcal{B}}$ alors il existe une suite de longueur $0+1$ constituée du terme $p = q_0 = p$ donc $p \equiv p$. Ainsi $\boxed{\equiv \text{ est réfléxive}}$.
   — Soit $(p, q) \in Q^2_{\mathcal{B}}$ tel que $p \equiv q$, par définition, il existe une suite de longueur $k + 1 \in \mathbb{N}^*$ constituée des termes $p = q_0, q_1, ..., q_k = q$, alors en considérant la suite constituée des termes $q = p_0, p_1, ..., p_{k-1}, p_k = p$, où $\forall\, 0 \leq j \leq k, p_j = q_{k-j}$. Ainsi on a :

$$\forall\, 0 \leq j < k,\ \varphi(p_{j+1}) = \varphi(q_{\underbrace{k-j-1}_{\in [\![1;k]\!]}}) = \varphi(q_{k-j}) = \varphi(p_j)$$

$$\text{ou } \psi(p_{j+1}) = \psi(q_{\underbrace{k-j-1}_{\in [\![1;k]\!]}}) = \psi(q_{k-j}) = \psi(p_j)$$

   Soit, $\forall\, 0 \leq j < k$, $\varphi(p_j) = \varphi(p_{j+1})$ ou $\psi(p_j) = \psi(p_{j+1})$
   Ce qui montre $p \equiv q \implies q \equiv p$, et donc que $\boxed{\equiv \text{ est symétrique}}$.

— Soit $(p,q,r) \in Q_{\mathcal{B}}^3$ tel que $p \equiv q$ et $q \equiv r$, alors il existe deux suites, de taille respective $l+1$ et $m+1$ ($(\ell, m) \in \mathbb{N}^2$), constituées des termes $p = q_0, q_1, ..., q_\ell = q$ et $q = r_0, r_1, ..., r_m = r$.
Alors en considérant la suite de taille $k+1$ (où $k = (\ell + m) \in \mathbb{N}$) constituée des termes $p = q_0, q_1, ..., q_\ell = r_0, r_1, ..., r_m = r$, on a clairement :

$$\forall\, 0 \leq j < k,\ \varphi(p_j) = \varphi(p_{j+1}) \text{ ou } \psi(p_j) = \psi(p_{j+1}) \text{(même lorsque } j = \ell \text{ puisque } \varphi(q_\ell) = \varphi(r_0) = \varphi(r_1))$$

Donc $(p \equiv q) \wedge (q \equiv r) \implies p \equiv r$, ce qui conclut sur la $\boxed{\text{transitivité de } \equiv}$.

On a ainsi montré que $\boxed{\equiv \text{ est une relation d'équivalence}}$.

23. Soit $(p,q) \in Q_{\mathcal{B}}^2$ tel que $p \equiv q$. Soit $p = q_0, q_1, ..., q_k = q$ la suite associée à $\equiv$.
Soit $\sigma \in \{a,b\}$, $j \in [\![0;k[\![$. Si $\varphi(q_j) = \varphi(q_{j+1})$, il découle :

$$\begin{aligned}
\varphi(\delta_{\mathcal{B}}(q_j,\sigma)) &= \delta_{\mathcal{A}}(\varphi(q_j),\sigma) && \text{par propriété de morphisme de } \varphi \\
&= \delta_{\mathcal{A}}(\varphi(q_{j+1}),\sigma) \\
&= \varphi(\delta_{\mathcal{B}}(q_{j+1},\sigma)) && \text{à nouveau par propriété de } \varphi
\end{aligned}$$

Le résultat est clairement similaire avec ψ, si on suppose $\psi(q_j) = \psi(q_{j+1})$.
Ainsi en définissant la suite $\Delta_0, \Delta_1, ..., \Delta_k$, avec $\forall j \in [\![0;k]\!],\ \Delta_j = \delta_{\mathcal{B}}(q_j,\sigma)$, alors

$$\forall\, 0 \leq j < k,\ \varphi(\Delta_j) = \varphi(\Delta_{j+1}) \text{ ou } \psi(\Delta_j) = \psi(\Delta_{j+1}) \text{ (d'après ce que l'on vient de montrer).}$$

D'où le résultat : $\boxed{\delta_{\mathcal{B}}(p,\sigma) \equiv \delta_{\mathcal{B}}(q,\sigma)}$.

24. Soit $(p,q) \in Q_{\mathcal{B}}^2$ tel que $p \equiv q$. Par symétrie de la relation $\equiv$, montrer que $p \in F_{\mathcal{B}} \implies q \in F_{\mathcal{B}}$ suffira à montrer l'équivalence.
Soit $p = q_0, q_1, ..., q_k = q$ la suite associée à $\equiv$.
Soit $j \in [\![0;k[\![$. Si $\varphi(q_j) = \varphi(q_{j+1})$, montrons que $q_j \in F_{\mathcal{B}} \implies q_{j+1} \in F_{\mathcal{B}}$:

$$q_j \in F_{\mathcal{B}} \implies \varphi(q_j) = \varphi(q_{j+1}) \in F_{\mathcal{A}} \implies q_{j+1} \in F_{\mathcal{B}} \text{ (par propriété de morphisme de } \varphi)$$

Le résultat est clairement similaire avec ψ, si on suppose $\psi(q_j) = \psi(q_{j+1})$.
Et donc, par une récurrence immédiate, $p \in F_{\mathcal{B}} \implies q \in F_{\mathcal{B}}$ et ainsi $\boxed{p \in F_{\mathcal{B}} \iff q \in F_{\mathcal{B}}}$

25. Construisons un tel automate $\mathcal{C}$:
   — Comme imposé par l'énoncé : $Q_{\mathcal{C}} = \{S_0, S_1, ..., S_{l-1}\}$;
   — L'état initial $i_{\mathcal{C}}$ est $[i_{\mathcal{B}}]$;
   — Comme montré à la question 23., si $\sigma \in \{a,b\},\ (p,q) \in Q_{\mathcal{B}}^2$, alors $[p] = [q] \implies [\delta_{\mathcal{B}}(p,\sigma)] = [\delta_{\mathcal{B}}(q,\sigma)]$.
   On peut donc définir $\forall \sigma \in \{a,b\},\ \forall [q] \in Q_{\mathcal{C}},\ \delta_{\mathcal{C}}([q],\sigma) = [\delta_{\mathcal{B}}(q,\sigma)]$;
   — De même, la question 24. montre que si $[p] = [q]$ alors $p \in F_{\mathcal{B}} \iff q \in F_{\mathcal{B}}$ donc on définit $F_{\mathcal{C}}$ ainsi : $F_{\mathcal{C}} = \{[q] \mid q \in F_{\mathcal{B}}\}$.

$\boxed{\text{On a ainsi construit } \mathcal{C} = \langle Q_{\mathcal{C}}, i_{\mathcal{C}}, \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle}$, montrons maintenant que η est bien un morphisme d'automates de $\mathcal{B}$ vers $\mathcal{C}$. Comme $\mathcal{B}$ et $\mathcal{C}$ sont deux automates accessibles, il suffit de montrer que η vérifie les conditions (2), (3) et (4) (question 16.) :
   (2) Par définition de $i_{\mathcal{C}}$, on a clairement $\eta(i_{\mathcal{B}}) = [i_{\mathcal{B}}] = i_{\mathcal{C}}$;
   (3) Soit $q \in Q_{\mathcal{B}},\ \sigma \in \{a,b\}$, Comme $\eta(q) = [q]$, on a clairement :

$$\eta(\delta_{\mathcal{B}}(q,\sigma)) = [\delta_{\mathcal{B}}(q,\sigma)] = \delta_{\mathcal{C}}([q],\sigma) = \delta_{\mathcal{C}}(\eta(q),\sigma) \text{ ;}$$

   (4) Enfin, encore une fois d'après la définition de $\mathcal{C}$: $\forall q \in Q_{\mathcal{B}},\ q \in F_{\mathcal{B}} \iff [q] = \eta(q) \in F_{\mathcal{C}}$;

$\boxed{\text{Donc } \eta \text{ est bien un morphisme d'automates de } \mathcal{B} \text{ vers } \mathcal{C}}$

26. Construisons un morphisme d'automates ψ' tel que $\psi' \circ \psi = \eta$.
Soit $q \in Q_{\mathcal{A}'}$, comme ψ est surjective, il existe $p \in Q_{\mathcal{B}}$ tel que $q = \psi(p)$ et donc on définit : $\psi'(q) = [p]$.
Supposons qu'il existe $p, p' \in Q_{\mathcal{B}}$ tels que $q = \psi(p) = \psi(p')$, alors $p \equiv p'$ et donc $[p] = [p']$, ce qui montre bien que $\psi'(q')$ ne dépend pas de l'antécédent pas ψ choisit, ce qui conclut bien sur la bonne définition de ψ'.
Montrons maintenant que ψ' est bien un morphisme d'automates. Comme $\mathcal{B}, \mathcal{A}'$ accessibles, il suffit de montrer que ψ' vérifie (2), (3) et (4) (question 16.) :

(2) Comme $i_{\mathcal{A}'} = \psi(i_{\mathcal{B}})$, $\psi'(i_{\mathcal{A}'}) = [i_{\mathcal{B}}] = i_{\mathcal{C}}$;

(3) Soit $q \in \mathcal{A}'$, $\sigma \in \{a, b\}$, il existe $p \in Q_{\mathcal{B}}$ tel que $q = \psi(p)$. On a alors :

$$\begin{aligned}
\delta_{\mathcal{A}'}(q,\sigma) &= \delta_{\mathcal{A}'}(\psi(p),\sigma) & \\
&= \psi(\delta_{\mathcal{B}}(p,\sigma)) & \text{par propriété de morphisme de } \psi \\
\text{Donc } \psi'(\delta_{\mathcal{A}'}(q,\sigma)) &= \underbrace{\psi' \circ \psi}_{=\eta}(\delta_{\mathcal{B}}(p,\sigma)) & \\
&= [\delta_{\mathcal{B}}(p,\sigma)] & \text{par définition de } \eta \text{ (cf question 25.)} \\
&= \delta_{\mathcal{C}}([p],\sigma) & \text{par définition de } \delta_{\mathcal{C}} \text{ ((cf question 25.))} \\
&= \delta_{\mathcal{C}}(\psi'(q),\sigma) & \text{car } \psi'(q) = [p] \text{ par construction}
\end{aligned}$$

(4) Soit $q \in Q_{\mathcal{A}'}$, il existe $p \in Q_{\mathcal{B}}$ tel que $q = \psi(p)$:

$$\begin{aligned}
q \in F_{\mathcal{A}'} &\iff \psi(p) \in F_{\mathcal{A}'} & \\
&\iff p \in F_{\mathcal{B}} & \text{par propriété de morphisme de } \psi \\
&\iff [p] \in F_{\mathcal{C}} & \text{par définition de } F_{\mathcal{C}} \text{ (cf question 25.)} \\
&\iff \psi'(q) \in F_{\mathcal{C}} & \text{car } \psi'(q) = [p] \text{ par construction}
\end{aligned}$$

Donc ψ', ainsi construit, est bien un morphisme d'automates de $\mathcal{A}'$ vers $\mathcal{C}$. En procédant de même avec φ', en remplacant $\mathcal{A}'$ par $\mathcal{A}$, on $\boxed{\text{construit bien } \varphi' \text{ et } \psi' \text{ deux tels morphismes}}$.

27.
```
let maxi_positif arr = (*renvoie l'element maximum d'un tableau d'entiers positifs
↪ (et -1 si la liste est vide)*)
    if Array.length arr = 0 then -1 else
    let value = ref arr.(0) in
    for i=1 to (Array.length arr) - 1 do
        value := max arr.(i) !value
    done;
    !value ;;

let renomme arr =
    let size = Array.length arr in
    let l = maxi_positif arr + 1 in (*entier l positif permettant de creer le
    ↪ dictionnaire (cf ci dessous)*)
    let index = Array.make l (-1) in (*semblant de dictionnaire qui a une valeur n
    ↪ associe une cle k (son indice dans le tableau renomme)*)
    let res = Array.make size 0 in (*tableau renomme*)
    let compteur = ref 0 in (*compteur permettant de garder en memoire la derniere
    ↪ cle associe a une valeur*)
    for i = 0 to size - 1 do
        begin
        if index.(arr.(i)) = -1 (*verifie que l'on a pas deja associe a la valeur n
        ↪ une cle*)
            then begin index.(arr.(i)) <- !compteur ;  (*on attribut a la valeur n
            ↪ une cle*)
                        incr compteur (*on incremente le compteur*)
                end ;
        res.(i) <- index.(arr.(i)) (*on renomme le tableau*)
        end
    done;
    res ;;
```

La fonction `maxi_positif` à une complexité linéaire en ℓ, et la fonction `renomme` effectue une boucle `for` comportant n étapes (où n est la taille du tableau donné en arguement). Les fonctions `Array.make` sont aussi de complexité linéaire en n pour `res` et ℓ pour `index`.
Ainsi la fonction $\boxed{\texttt{renomme} \text{ a une complexité en } O(\max(\ell, n))}$.

28.
```
let relation morph1 morph2 =
    let q_depart = Array.length morph1 in (*cardinal de l'automate de depart*)
    let q_arrive1 = maxi_positif morph1 + 1 in (*cardinal de l'automate image de
    ↪ phi*)
```

```
let q_arrive2 = maxi_positif morph2 + 1 in (*cardinal de l'automate image de psi*)
let voisins1 = Array.make q_arrive1 [] in (*voisins1.(q) = liste des antecedents de q par phi*)
let voisins2 = Array.make q_arrive2 [] in (*voisins2.(q) = liste des antecedents de q par psi*)
for i = 0 to (q_depart - 1) do (*on remplit les voisins*)
  voisins1.(morph1.(i)) <- i :: voisins1.(morph1.(i)) ;
  voisins2.(morph2.(i)) <- i :: voisins2.(morph2.(i))
done;
let morph_res = Array.make q_depart (-1) in (*initialisation de eta*)
let rec aux e arr = function (*fonction auxiliaire qui pour tout i dans la liste va attribuer a arr.(i) la valeur e*)
  |[] -> ()
  |h::t -> morph_res.(h) <- e ; aux e arr t in
for i = 0 to (q_depart - 1) do (*on remplit eta*)
  if morph_res.(i) = -1 then (*verifie si l'etat i ne s'est pas deja vu etre attribue une classe d'equivalence ...*)
    begin (*... si non, alors on initialise la classe d'equivalence (en lui donnant un numero quelconque (MAIS QUI N'A PAS ENCORE ETE DONNE))*)
      aux i morph_res voisins1.(morph1.(i)) ; (*on attribut cette classe d'equivalence aux voisins de q par phi*)
      aux i morph_res voisins2.(morph2.(i)) ; (*on attribut cette classe d'equivalence aux voisins de q par psi*)
    end
  else begin (*... si on a deja attribue une classe d'equivalence a cet etat, alors on l'attribut aux voisins de cet etat*)
    aux morph_res.(i) morph_res voisins1.(morph1.(i)) ;
    aux morph_res.(i) morph_res voisins2.(morph2.(i)) ;
  end
  done;
  renomme morph_res ;; (*on renomme les classes d'equivalence (notamment afin d'avoir morph_res.(0) = 0)*)
```

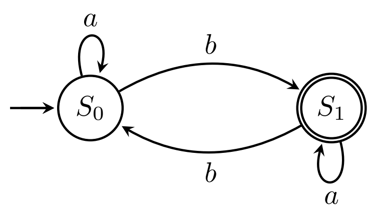
5 Réduction d'automates

5.1 Existence et unicité

29. D'après les questions 21. et 25, on peut $\boxed{\text{construire un automate } \mathcal{C}}$ à partir de la partie accessible du produit d'automates $\mathcal{A} \times \mathcal{A}'$.
La question 26. assure $\boxed{\text{l'existence de deux morphismes } \varphi' : \mathcal{A} \to \mathcal{C} \text{ et } \psi' : \mathcal{A}' \to \mathcal{C}}$.

30. L'automate $\mathcal{A} \times \mathcal{A}'$ possède 2 classes d'équivalence pour la relation $\equiv$: $(E, H) \equiv (F, H) \equiv (F, I)$ et $(G, J) \equiv (G, K)$. On les note respectivement S_0 et S_1.
La représentation de $\mathcal{C}$ est donc :

Automate $\mathcal{C}$



$\varphi' : \mathcal{A} \to \mathcal{C}$ est représentée par

q	$\varphi'(q)$
E	S_0
F	S_0
G	S_1

et ψ' est représentée par

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

31. Soit $\mathcal{A}, \mathcal{A}' \in \mathfrak{R}_L$ (i.e $L = L(\mathcal{A}) = L(\mathcal{A}')$) tels que $|Q_{\mathcal{A}}| = |Q_{\mathcal{A}'}| = m_L$.

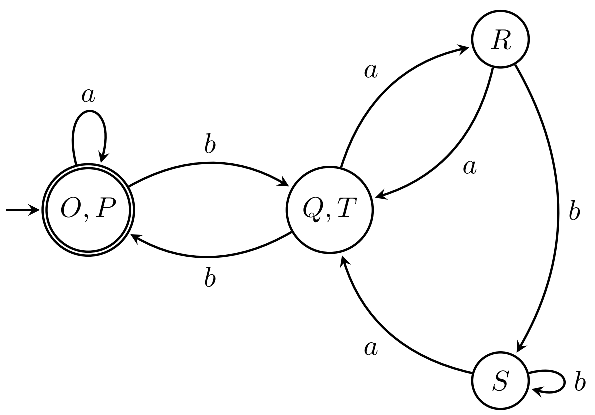
Par le même procédé qu'à la question 29., on construit $\mathcal{C}$, $\varphi' : \mathcal{A} \to \mathcal{C}$ et $\psi' : \mathcal{A}' \to \mathcal{C}$.
Par surjectivité de φ' et ψ' : $|Q_\mathcal{C}| \leq m_L$. De plus l'existence de φ' et ψ' assure que $L(\mathcal{C}) = L(\mathcal{A}) = L(\mathcal{A}')$.
Donc $\mathcal{C} \in \mathfrak{R}_L$, d'où $Q_\mathcal{C}| \geq m_L$ (par défintion de m_L). Donc $|Q_\mathcal{C}| = |Q_\mathcal{A}| = |Q_{\mathcal{A}'}|$.
Donc, d'après la question 14., φ' et ψ' sont des isomorphismes d'automates.
Toujours d'après la question 14, ψ'^{-1} est aussi un isomorphisme d'automates, et donc comme composition d'isomorphismes : $\psi'^{-1} \circ \varphi'$ est un isomorphisme de $\mathcal{A}$ vers $\mathcal{A}'$ (cf question 15.).
$\boxed{\text{Donc } \mathcal{A} \text{ et } \mathcal{A}' \text{ sont nécessairement isomorphes}}$.

32. Soit $\mathcal{A}, \mathcal{M}_L \in \mathfrak{R}_L$ (i.e $L = L(\mathcal{A}) = L(\mathcal{M}_L)$) tels que $|Q_{\mathcal{M}_L}| = m_L$.
Par le même procédé qu'à la question 29., on construit $\mathcal{C}$, $\varphi' : \mathcal{A} \to \mathcal{C}$ et $\psi' : \mathcal{M}_L \to \mathcal{C}$.
Par surjectivité de ψ' : $|Q_\mathcal{C}| \leq m_L$. De plus l'existence de φ' et ψ' assure que $L(\mathcal{C}) = L(\mathcal{A}) = L(\mathcal{A}')$. Donc $\mathcal{C} \in \mathfrak{R}_L$, d'où $Q_\mathcal{C}| \geq m_L$ (par défintion de m_L). Donc $|Q_\mathcal{C}| = |Q_{\mathcal{M}_L}|$.
Donc, d'après la question 14., ψ' est un isomorphisme d'automates.
Toujours d'après la question 14, ψ'^{-1} est aussi un isomorphisme d'automates, et donc comme composition de morphismes : $\varphi = \psi'^{-1} \circ \varphi'$ est un morphisme de $\mathcal{A}$ vers $\mathcal{M}_L$ (cf question 15.).
$\boxed{\text{Donc il existe bien un morphisme } \varphi : \mathcal{A} \to \mathcal{M}_L}$.

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

33.

Automate $\mathcal{A}_6^{O,P}$



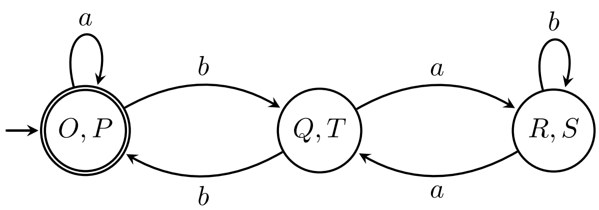
Afin de pouvoir créer un morphisme φ de $\mathcal{A}_6$ vers $\mathcal{A}_6^{O,P}$, il faut aussi que l'on fusionne les états Q et T afin d'avoir les bonnes transitions par b. On peut alors représenter φ par

q	$\psi'(q)$
O	O, P
P	O, P
Q	Q, T
R	R
S	S
T	Q, T

. Ainsi, on a bien $\varphi(O) = \varphi(P)$ et $|Q_{\mathcal{A}_6^{O,P}}| < |Q_{\mathcal{A}_6}|$

34. Les transitions des états Q et R selon b dans $\mathcal{A}_6$ sont respectivment $O \in F_{\mathcal{A}_6}$ et $S \notin F_{\mathcal{A}_6}$. Ainsi par les propriétés (3) et (4) de morphisme de ψ, $\boxed{\text{il n'est pas possible d'avoir un tel } \mathcal{A}_6^{Q,R}}$.

35. En partant de l'automate $\mathcal{A}_6^{O,P}$, on peut encore fusionner les états R et S.
$\boxed{\text{On obtient alors un automate } \mathcal{M}_{L_6} \text{ à trois états qui reconnait le même langage que } \mathcal{A}_6}$:



Automate $\mathcal{M}_{L_6}$

36.
```
let table_de_predecesseurs aut =
    let n, delta, f  = aut in
    let tab = Array.make_matrix n n false in (*on initalise le tableau*)
```

```
    let rec aux p q = (*fonction auxiliaire qui visite tous les (p, q) pour un
    ↪  couple (p0, q0) donnee*)
    if not tab.(p).(q)  (*s'assure de ne visiter les sommets qu'une seule fois*)
    then begin
        tab.(p).(q) <- true ;
        let succP_a, succP_b = delta.(p) in
        let succQ_a, succQ_b = delta.(q) in
        aux succP_a succQ_a ; (*début du parcours*)
        aux succP_b succQ_b ; (*suite du parcours (en profondeur)*)
        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)) (*pour tous les
            ↪  couples (p, q), on regarde si il respecte les conditions de
            ↪  l'enonce ...*)
            then aux p q (*... si oui, alors on le visite*)
        done;
    done;
    tab ;;
```

37. On considère le graphe orienté P' similaire à P, mais où l'orientation des arcs est inversée (i.e que pour tout $(p,q) \in Q \times Q$ et pour toute lettre $\sigma \in \{a,b\}$ les arcs vont de $(\delta(p,\sigma), \delta(q,\sigma))$ vers (p,q)). En faisant une fonction `table_de_predecesseurs_bis`, simillaire à la fonction de la question 36. mais pour le graphe P', on aura alors `tab.(p).(q) = false` si et seulement si pour tout mot $m \in L(\mathcal{A})$, $(\delta^*_{\mathcal{A}}(p,m) \in F_{\mathcal{A}}$ et $\delta^*_{\mathcal{A}}(q,m) \in F_{\mathcal{A}})$ ou $(\delta^*_{\mathcal{A}}(p,m) \notin F_{\mathcal{A}}$ et $\delta^*_{\mathcal{A}}(q,m) \notin F_{\mathcal{A}})$. On dira que p est en relation avec q (et on notera cette relation $\star$).
Cette relation est clairement refléxive, symétrique et transitive, $\star$ est donc une relation d'equivalence. Ainsi pour réduire l'automate $\mathcal{A}$ on va déterminer les classes d'équivalence de la relation $\star$, et fusionner les états appartenant à chacune des classes. Pour ce faire, on determinera un morphisme φ entre $\mathcal{A}$ et l'automate dont les états sont les classes d'équivalence de $\star$ (même procédé qu'à la question 26.), pour ensuite calculer l'image de $\mathcal{A}$ par φ pour $\boxed{\text{obtenir } \mathcal{M}_L}$.

```
let table_de_predecesseurs_bis aut =
    let n, delta, f = aut in
    let tab = Array.make_matrix n n false in
    let transi = Array.make_matrix n n [] in (*graphe P'*)
    for p = 0 to n - 1 do (*construction de P'*)
        for q = 0 to n - 1 do
            let succP_a, succP_b = delta.(p) in
            let succQ_a, succQ_b = delta.(q) in
            transi.(succP_a).(succQ_a) <- (p, q) :: transi.(succP_a).(succQ_a);
            transi.(succP_b).(succQ_b) <- (p, q) :: transi.(succP_b).(succQ_b);
        done;
    done;
    let rec aux p q = (*fonction auxiliaire qui parcours P' pour mettre a jour les
    ↪  relations "etoiles" entre les etats*)
        if not tab.(p).(q)
        then begin
            tab.(p).(q) <- true ;
            List.iter (fun (i,j) -> aux i j) transi.(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 aux p q
        done;
    done;
    tab ;;
```

```
let etoile aut = (*meme procede que pour `relation` mais cette fois-ci on veut le
↪  morphisme et un etat quelconque dans chacune des classes*)
    let n, delta, f = aut in
    let pred = table_de_predecesseurs_bis aut in
    let classes_equiv = Array.make n (-1) in
    for p = 0 to n - 1 do
        if classes_equiv.(p) = -1
        then begin
            classes_equiv.(p) <- p ;
            for q = 0 to n - 1 do
                if not pred.(p).(q)
                then classes_equiv.(q) <- p
            done;
            end
    done;
    let classes_equiv = renomme classes_equiv in
    let nb_classes = maxi_positif classes_equiv + 1 in
    let etats_classe = Array.make nb_classes (-1) in
    Array.iteri (fun i elem -> etats_classe.(elem) <- i) classes_equiv ;
    classes_equiv, etats_classe ;;

let reduit aut =
    let n, delta, f = aut in
    let classes_equiv, etats_classe = etoile aut in
    let new_n = Array.length etats_classe in
    let new_delta = Array.make new_n (-1, -1) in
    let new_f = Array.make new_n false in
    for p = 0 to new_n - 1 do
        let succ_a, succ_b = delta.(etats_classe.(p)) in
        new_delta.(p) <- classes_equiv.(succ_a), classes_equiv.(succ_b) ;
        new_f.(p) <- f.(etats_classe.(p))
    done ;
    (new_n, new_delta, new_f) ;;
```

Exemple : en testant ces fonctions avec $\mathcal{A}_6$, on retrouve bien $\mathcal{M}_{L_6}$:

```
INPUT :
let a6 = 6, [|(0, 2) ; (0, 5); (3, 1); (2, 4); (5, 4); (3, 0)|], [|true ; true ; false
↪  ; false ; false ; false|] in
reduit a6 ;;

OUTPUT :
- : int * (int * int) array * bool array =
(3, [|(0, 1); (2, 0); (1, 2)|], [|true; false; false|])
```

FIN DE LA COPIE