DS 7 d'option informatique - MP* - Sujet Centrale 2022 - Correction

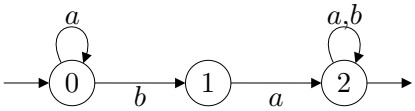
Question 1 – Un mot $u \in \{a,b\}^\star$ est accepté par $\mathcal{A}_1$ si et seulement s'il existe un chemin de l'état 0 à l'état 2 étiqueté par u. La forme de $\mathcal{A}_1$ permet de décomposé u comme une concaténation $u = u_1 \cdot a \cdot b \cdot u_2$ où $u_1 \in \{a,b\}^\star$ et $u_2 \in \{a\}^\star$. Ainsi :

$L_1 = \{a,b\}^\star \cdot \{ab\} \cdot \{a\}^\star$. C'est l'ensemble des mots u de $\{a,b\}^\star$ tels que u contient au moins une fois la lettre b et tel que le dernier b de u soit précédé d'un a.

Un mot u de $\tilde{L}_1$ s'écrit donc comme une concaténation $u = v_1 \cdot b \cdot a \cdot v_2$ où $v_1 \in \{a\}^\star$ et $v_2 \in \{a,b\}^\star$. Ainsi :

$\tilde{L}_1 = \{a\}^\star \cdot \{ba\} \cdot \{a,b\}^\star$. C'est l'ensemble des mots u de $\{a,b\}^\star$ tels que u contient au moins une fois la lettre b et tel que le premier b de u soit suivi d'un a.

Question 2 – L'automate suivant convient :



Question 3 – À partir d'un automate $A = (Q, I, F, T)$, on définit l'automate miroir $\tilde{A} = (Q, I', F', T')$ par :

$$I' = F \qquad F' = I \qquad T' = \left\{(q,a,q') \in Q \times \Sigma \times Q : (q',a,q) \in T\right\}$$

En d'autres termes, les état initiaux (resp. finaux) de $\tilde{A}$ sont les états finaux (resp. initiaux) de A. De plus, les directions des arcs dans $\tilde{A}$ ont été inversées par rapport à A.

Par construction, pour tout $(q,a,q') \in Q \times \Sigma \times Q$, la transition $q \xrightarrow{a} q'$ apparaît dans A si et seulement si la transition $q' \xrightarrow{a} q$ apparaît dans $\tilde{A}$. Ainsi, pour tout $(q_0, \dots, q_n) \in Q^{n+1}$ et tout $(a_0, \dots, a_{n-1}) \in \Sigma^n$:

$$q_0 \in I,\ q_n \in F \text{ et } q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots q_{n-1} \xrightarrow{a_{n-1}} q_n \text{ dans } A.$$

$$\Leftrightarrow$$

$$q_n \in I',\ q_0 \in F' \text{ et } q_n \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_{n-2}} \dots q_1 \xrightarrow{a_0} q_0 \text{ dans } \tilde{A}.$$

On en déduit que pour tout $w \in \Sigma^\star$:

$$w \text{ est reconnu par } A. \Leftrightarrow \tilde{w} \text{ est reconnu par } \tilde{A}.$$

Question 4 –

```
let transpose (a: automate): automate = {
    nb = a.nb;
    init = a.final;
    final = a.init;
    trans = List.map (fun (q1, a, q2) -> (q2, a, q1)) a.trans;
  };;
```

Question 5 – Dans la fonction `transpose`, la création des champs `nb`, `init` et `final` se fait en temps constant. Pour créer le champ `trans`, la fonction `List.map` parcourt tous les éléments de `a.trans`, ce qui prend un temps $\mathcal{O}(m)$ où m est le nombre de transitions dans A. Finalement :

Le temps d'exécution de `transpose` est en $\mathcal{O}(m)$ avec m le nombre de transitions dans A.

Question 6 –

```
let palindrome (s: string): bool =
  let n = String.length s in
  let rec verif i =
    if i >= n-1-i then true
    else s.[i] = s.[n-1-i] && verif (i+1)
  in
  verif 0;;
```

Question 7 – Supposons que Σ soit un alphabet à une lettre notée a. Tous les mots sont de la forme a^n avec $n \in \mathbb{N}$ et donc tous les mots sont des palindromes. Ainsi :

$$\boxed{\text{Pal}(\Sigma) = \{a\}^\star \text{ qui est donc rationnel.}}$$

Question 8 – Soit Σ un alphabet contenant au moins deux lettres notées a et b, et supposons par l'absurde que $\text{Pal}(\Sigma)$ est rationnel. D'après le lemme de l'étoile, il existe un entier $n \in \mathbb{N}^\star$ tel que pour tout mot $u \in \text{Pal}(\Sigma)$ vérifiant $|u| \geqslant n$, il existe des mots x, y, z tels que :

$$u = xyz \qquad y \neq \varepsilon \qquad |xy| \leqslant n \qquad \forall k \in \mathbb{N} : xy^k z \in \text{Pal}(\Sigma).$$

Soit $n \in \mathbb{N}^\star$ l'entier donné par le lemme et posons $u = a^n b a^n$. On remarque que $u \in \text{Pal}(\Sigma)$ et $|u| \geqslant n$. Puisque les trois mots x, y, z vérifient $u = xyz$ et $|xy| \leqslant n$, on a $y = a^m$ avec $m > 0$. Ainsi pour $k = 0$, on obtient $xy^0 z = a^{n-m} b a^n \in \text{Pal}(\Sigma)$, ce qui constitue une contradiction.

Question 9 – Si on définit l'automate $A' = (Q, \{q\}, \{q'\}, T)$, alors un mot $w = a_0 \dots a_{n-1}$ est reconnu par A' si et seulement s'il existe une succession de transitions :

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots q_{n-1} \xrightarrow{a_{n-1}} q_n \qquad \text{avec} \qquad q_0 = q \text{ et } q_n = q'.$$

Comme les transitions dans A et A' sont identiques, on obtient :

$$\boxed{L_{A'} = L_{q,q'} \text{ qui est bien reconnaissable.}}$$

Le langage L_A est l'ensemble des mots qui étiquettent un chemin dans A partant d'un état $q \in I$ et arrivant en un état $q' \in F$. Ainsi :

$$\boxed{L_A = \bigcup_{q \in I} \bigcup_{q' \in F} L_{q,q'}.}$$

Question 10 – Montrons le par double inclusion.

($\supset$) Soit $u \in \Sigma^\star$ et $w = u\tilde{u}$, alors :

$$|u\tilde{u}| = |u| + |\tilde{u}| = 2|u| \qquad \text{donc} \qquad u \in (\Sigma^2)^\star$$

De plus :

$$\tilde{w} = \widetilde{u\tilde{u}} = \tilde{\tilde{u}}\tilde{u} = u\tilde{u} = w \qquad \text{donc} \qquad u \in \text{Pal}(\Sigma)$$

($\subset$) Soit $w \in \text{Pal}(\Sigma) \cap (\Sigma^2)^\star$. Comme $|w|$ est pair, on peut écrire $w = uv$ avec $|u| = |v| = |w|/2$. De plus :

$$uv = w = \tilde{w} = \tilde{v}\tilde{u}$$

Comme $|\tilde{u}| = |\tilde{v}|$, on obtient $u = \tilde{v}$ et $v = \tilde{u}$ et donc $w = u\tilde{u}$.

Question 11 – Les éléments de $\mathscr{L}(a^\star b)$ sont de la forme $a^n b$ avec $n \in \mathbb{N}$, ainsi :

$$\boxed{D(a^\star b) = \{a^n b b a^n : n \in \mathbb{N}\}.}$$

Montrons par double inclusion que :

$$\boxed{R(a^\star b^\star a^\star) = \mathscr{L}(a^\star b^\star).}$$

($\subset$) Soit $w \in R(a^\star b^\star a^\star)$, alors $w\tilde{w} \in \mathscr{L}(a^\star b^\star a^\star)$. Le mot $v = w\tilde{w}$ est donc de la forme $v = a^i b^j a^k$ avec $(i,j,k) \in \mathbb{N}^3$ et d'après la question précédente, v est un palindrome de taille paire. Ainsi :

$$a^i b^j a^k = v = \tilde{v} = a^k b^j a^i$$

Si $j = 0$, alors w est un préfixe de a^{i+k}, donc $w \in \mathscr{L}(a^\star b^\star)$. Sinon, à partir de l'égalité ci-dessus, on obtient $i = k$. Comme $|v| = 2i + j$ est pair, j est pair et w est égal à la première moité de $v = w\tilde{w}$:

$$w = a^i b^{j/2} \in \mathscr{L}(a^\star b^\star).$$

($\supset$) Si $w = a^n b^m$ avec $(n,m) \in \mathbb{N}^2$, alors :

$$w\tilde{w} = a^n b^{2m} a^n \in \mathscr{L}(a^\star b^\star a^\star) \qquad \text{donc} \qquad w \in R(a^\star b^\star a^\star)$$

Question 12 – On remarque que pour tout mot w :

$$\begin{aligned} w \in R(L) &\Leftrightarrow w\tilde{w} \in L \\ &\Leftrightarrow \exists q_1 \in I, \exists q_2 \in Q, \exists q_3 \in F : w \in L_{q_1,q_2} \text{ et } \tilde{w} \in L_{q_2,q_3}. \\ &\Leftrightarrow \exists q_1 \in I, \exists q_2 \in Q, \exists q_3 \in F : w \in L_{q_1,q_2} \text{ et } w \in \tilde{L}_{q_2,q_3}. \\ &\Leftrightarrow w \in \bigcup_{q_1 \in I} \bigcup_{q_2 \in Q} \bigcup_{q_3 \in F} L_{q_1,q_2} \cap \tilde{L}_{q_2,q_3} \end{aligned}$$

Le langage situé à droite de l'équivalence est défini par des unions et intersections finies de langages rationnels (voir les questions 9 et 3). Par les propriétés de stabilités de l'ensemble de langages rationnels, on obtient :

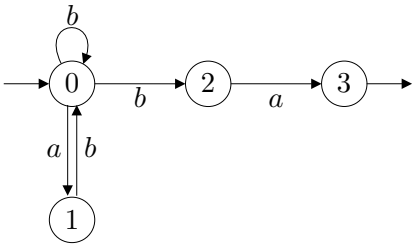
Si L est reconnaissable, alors $R(L)$ est reconnaissable.

En revanche :

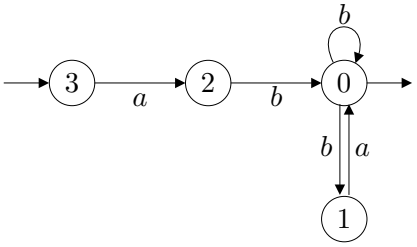
En fonction de L, le langage $D(L)$ peut être reconnaissable ou non.

Par exemple, si L est le langage rationnel $L = \mathscr{L}(a^\star)$, alors $D(L) = \mathscr{L}((aa)^\star)$ qui est rationnel donc reconnaissable. En revanche, d'après la question 11, si $L = \mathscr{L}(a^\star b)$, alors $D(L) = \{a^n bba^n : n \in \mathbb{N}\}$ qui n'est pas rationnel par le lemme de l'étoile (preuve similaire à celle de la question 8 en remplaçant $a^n ba^n$ par $a^n bba^n$).

Question 13 – L'automate suivant convient :

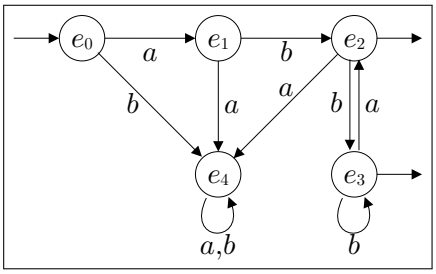


Question 14 – L'automate miroir $\widetilde{\mathcal{A}_2}$ est :



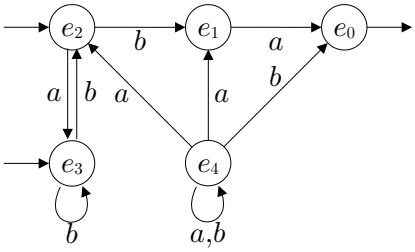
On calcule d'abord la table des transitions de $\widetilde{\mathcal{A}_2}$, puis on en déduit $\mathcal{A}_3$:

	a	b
$e_0 = \{3\}$	$\{2\}$	$\{\}$
$e_1 = \{2\}$	$\{\}$	$\{0\}$
$e_2 = \{0\}$	$\{\}$	$\{0,1\}$
$e_3 = \{0,1\}$	$\{0\}$	$\{0,1\}$
$e_4 = \{\}$	$\{\}$	$\{\}$



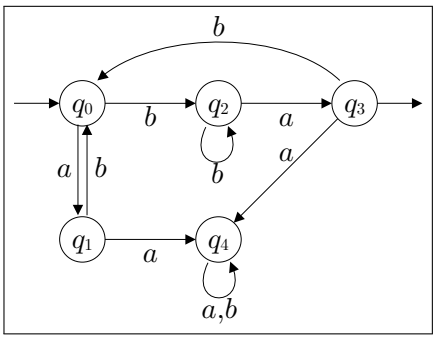
Remarque. L'état e_4 est un état puits non co-accessible qu'on peut supprimer sans modifier le langage reconnu par l'automate.

Question 15 – L'automate miroir $\widetilde{\mathcal{A}_3}$ est :



On calcule d'abord la table des transitions de $\widetilde{\mathcal{A}_3}$, puis on en déduit $\mathcal{A}_4$:

	a	b
$q_0 = \{e_2, e_3\}$	$\{e_3\}$	$\{e_1, e_2, e_3\}$
$q_1 = \{e_3\}$	$\{\}$	$\{e_2, e_3\}$
$q_2 = \{e_1, e_2, e_3\}$	$\{e_0, e_3\}$	$\{e_1, e_2, e_3\}$
$q_3 = \{e_0, e_3\}$	$\{\}$	$\{e_2, e_3\}$
$q_4 = \{\}$	$\{\}$	$\{\}$



Question 16 – À l'aide de la question 3, on remarque que :

→ $\widetilde{\mathcal{A}_2}$ et $\mathcal{A}_3$ reconnaissent $\tilde{L}_2$

→ $\widetilde{\mathcal{A}_3}$ et $\mathcal{A}_4$ reconnaissent $\tilde{\tilde{L}}_2 = L_2$

Donc :

Le langage reconnu par $\mathcal{A}_4$ doit être $L_2 = (b + ab)^\star ba$

Question 17 –

```
let rec supprimer (li: 'a list): 'a list = match li with
  | [] -> []
  | e :: q -> e :: supprimer (List.filter (fun x -> x <> e) q);;
```

Question 18 – Lorsqu'on appelle la fonction `supprimer` sur une liste non vide `e::q` de taille n :

→ La fonction `List.filter` s'exécute en temps $\mathcal{O}(n)$ car elle parcourt tous les éléments de `q`.

→ La fonction `supprimer` est appelée récursivement sur la liste renvoyée par `List.filter`. Ainsi dans le pire cas, aucun élément n'est supprimé et l'appel récursif se fait sur une liste de taille $n - 1$.

On a donc une complexité de la forme :

$$C_n = \sum_{k=0}^{n} T_k \qquad \text{avec} \qquad T_k = \mathcal{O}(k)$$

Le temps d'exécution est en $\mathcal{O}(n^2)$ où n est la taille de la liste en entrée.

Question 19 – On remarque que pour tout $q \in [\![0; n-1]\!]$:

$$\sum_{i=0}^{q-1} 2^i = 2^q - 1 < 2^q \qquad \text{donc} \qquad \left\lfloor \frac{\text{numero(X)}}{2^q} \right\rfloor = \sum_{i \in X \cap [\![q; n-1]\!]} 2^{i-q}$$

De plus, pour tout $i \in [\![q; n-1]\!]$:

$$2^{i-q} \equiv \begin{cases} 1 \mod 2 \text{ si } i = q \\ 0 \mod 2 \text{ si } i > q \end{cases}$$

Ainsi :

$$\left\lfloor \frac{\text{numero(X)}}{2^q} \right\rfloor \equiv 1 \mod 2 \Leftrightarrow q \in X$$

```
let est_dans (q: int) (k: int): bool =
  (k / pow.(q)) mod 2 = 1;;
```

Question 20 –

```
let rec numero (li: int list): int = match li with
  | [] -> 0
  | i :: q -> let k = numero q in
              if est_dans i k then k else k + pow.(i);;
```

Question 21 –

```
let rec intersecte (li: int list) (k: int): bool = match li with
  | [] -> false
  | q :: li2 -> est_dans q k || intersecte li2 k;;
```

Question 22 –

```
(* Cette fonction renvoie deux listes représentant les états \delta(X, a) et
 * \delta(X, b) *)
let rec liste_etat_suivant k t: int list * int list = match t with
  | [] -> [], []
  | e :: q -> let li1, li2 = liste_etat_suivant k q in
              match e with
              | (q1, _, _) when not (est_dans q1 k) -> li1, li2
              | (_, 'a', q2) -> q2 :: li1 , li2
              | (_, 'b', q2) -> li1 , q2 :: li2
              | _ -> failwith "Seules les lettres a et b sont autorisées";;

let etat_suivant (k: int) (t: (int*char*int) list): int * int =
  let li1, li2 = liste_etat_suivant k t in
  numero li1, numero li2;;
```

Question 23 –

```
let rec cherche (k0: int) (li: (int*int) list): int = match li with
  | [] -> -1
  | (k,v) :: q when k0 = k -> v
  | _ :: q -> cherche k0 q;;
```

Question 24 – Pour construire le déterministe accessible A_{det}, on utilise un parcours en profondeur de A_{det} (dans ces explications, on identifie A_{det} et le graphe qui lui est associé) en partant du sommet initial de A_{det}.

→ Les deux listes `trans` et `final` sont initialement vides et sont complétées lors de la découverte de nouveaux états par le parcours en profondeur.

→ Il ne faut pas oublier de renommer les état comme expliqué par l'énoncé juste avant la question 23. Pour cela, lors du parcours, on construit également une liste `li` contenant les couples (k, v).

→ Le parcours en profondeur est effectué par une fonction récursive `explorer` qui prend en entrée le numéro k d'un ensemble d'états et explore le sommet de A_{det} correspondant à k. La fonction `explorer` renvoie également le numéro v associé à k.

```
1  let determinise (a: automate): automate =
2    let nb = ref 0 in
3    let li: (int*int) list ref = ref [] in (* décrit Q23 *)
4    let trans = ref [] in
5    let final = ref [] in
6    let rec explorer (k: int): int = match cherche k !li with
7      | v when v <> -1 -> v
8      | _ -> let v = !nb in
9             incr nb;
10            li := (k,v) :: !li;
11            if intersecte a.final k then final := v :: !final;
12            let ka, kb = etat_suivant k a.trans in
13            let va = explorer ka in
14            let vb = explorer kb in
15            trans := (v, 'a', va) :: (v, 'b', vb) :: !trans;
16            v
17   in
18   let init = [explorer (numero a.init)] in
19   {nb = !nb; init = init; final  = !final; trans = !trans};;
```

Question 25 – On remarque que l'arbre des appels récursifs de `explorer` est un arbre binaire strict. En effet, un appel correspond à un cas de base (ligne 7) ou donne lieu à deux appels récursifs (lignes 13 et 14). De plus, grâce à l'utilisation de la liste `li`, le cas de la ligne 8 est vérifié exactement N fois. En résumé, l'arbre des appels récursifs est composé de N noeuds internes et $N+1$ feuilles (car il est binaire strict). Sans compter les appels récursifs, lors de l'exécution de la fonction `explorer`, on a :

→ Un appel à la fonction `cherche` en temps $\mathcal{O}(\texttt{len(li)}) = \mathcal{O}(N)$.

→ Un appel à la fonction `intersecte` en temps $\mathcal{O}(\texttt{len(a.final)}) = \mathcal{O}(n)$.

→ Un appel à la fonction `etat_suivant` en temps $\mathcal{O}(\texttt{len(a.trans)})$. Dans le pire des cas, pour chaque paire d'états (q_1, q_2) de A et chaque lettre $c \in \Sigma$, on a une transition de q_1 à q_2 étiquetée par c. Comme $\Sigma = \{a, b\}$, le temps d'exécution est en $\mathcal{O}(\texttt{len(a.trans)}) = \mathcal{O}(n^2)$.

En conclusion :

La complexité de `determinise` est en $\mathcal{O}(N \times (N + n^2))$.

Question 26 – Soient $q \in Q$ et $u \in \Sigma^\star$ tels que $q \in \delta^\star(\{I\}, u)$. Comme δ est la fonction de transition du déterminé accessible de A, alors dans A, il existe un chemin étiqueté par u d'un état initial $q_0 \in I$ à q.

De plus, comme $\tilde{A}$ est accessible, dans $\tilde{A}$ il existe un chemin de l'état initial f à q. Ainsi, dans A, il existe un chemin de q à f étiqueté par un mot noté w.

Finalement dans A, il existe un chemin étiqueté par uw de q_0 à f passant par q. Donc $uw \in L$.

Question 27 – Supposons $u^{-1}L = v^{-1}L$ et montrons que $\delta^\star(\{I\}, u) \subset \delta^\star(\{I\}, v)$ (l'autre inclusion se montre en échangeant les rôles de u et v). Soit $q \in \delta^\star(\{I\}, u)$. En utilisant la question précédente, il existe un chemin γ dans A de q à f où γ est étiqueté par un mot $w \in \Sigma^\star$ tel que $uw \in L$. On a alors $w \in u^{-1}L = v^{-1}L$, ce qui signifie que $vw \in L$.

Si on s'intéresse à l'automate $\tilde{A}$, il existe un chemin $\tilde{\gamma}$ de f à q étiqueté par $\tilde{w}$. Étant donnée que $\tilde{A}$ est supposé déterministe, $\tilde{\gamma}$ est le seul chemin étiqueté par $\tilde{w}$ partant de f. De plus $\tilde{w}\tilde{v} \in \tilde{L}$, ce qui signifie que le seul chemin étiqueté par $\tilde{v}$ partant de q dans $\tilde{A}$ a pour état final un élément de I noté q_0. Ainsi, dans A, il existe un chemin entre q_0 et q étiqueté par v. On obtient bien $q \in \delta^\star(\{I\}, v)$

Question 28 – Soit A un automate quelconque reconnaissant L, alors $\tilde{A}$ et $B = (\tilde{A})_{\text{det}}$ reconnaissent $\tilde{L}$ et donc $\tilde{B}$ reconnaît L. D'où :

$\boxed{(\tilde{B})_{\text{det}} \text{ reconnaît } L.}$

Pour montrer que $(\tilde{B})_{\text{det}}$ vérifie la propriété $(*)$, on remarque que $\tilde{\tilde{B}} = B = (\tilde{A})_{\text{det}}$ est déterministe et accessible. Ainsi, en remplaçant A par $\tilde{B}$ dans le résultat de la question 27 :

$\boxed{(\tilde{B})_{\text{det}} \text{ vérifie la propriété } (*)}$

Question 29 –

```
let minimal (a: automate): automate =
  let b = determinise (transpose a) in
  determinise (transpose b);;
```

Question 30 –

```
let rec lettre (expr: exprat): int = match expr with
  | Vide -> 0
  | Epsilon -> 0
  | Lettre _ -> 1
  | Union (e1, e2) -> lettre e1 + lettre e2
  | Concat (e1, e2) -> lettre e1 + lettre e2
  | Etoile e -> lettre e ;;
```

Question 31 –

```
let rec est_vide (e: exprat): bool = match e with
  | Vide -> true
  | Epsilon -> false
  | Lettre _ -> false
  | Union (e1, e2) -> est_vide e1 && est_vide e2
  | Concat (e1, e2) -> est_vide e1 || est_vide e2
  | Etoile _ -> false;;
```

Question 32 –

```
let se (expr: exprat): exprat = match expr with
  | Etoile Vide -> Epsilon
  | Etoile Epsilon -> Epsilon
  | Etoile (Etoile e) -> Etoile e
  | _ -> expr;;
```

Question 33 – On remarque que :

→ Pour $n = 0$, on a $E_0 = a + \varnothing$. Il suffit d'appliquer une fois la règle $E + \varnothing \equiv E$ pour obtenir $E_0 \equiv a$.

→ Pour $n \geqslant 1$, lorsqu'on applique une fois la règle $E \cdot \varnothing \equiv \varnothing$, on obtient $E_n \equiv E_{n-1}$.

Par une récurrence immédiate sur n :

Pour tout $n \in \mathbb{N}$, $n+1$ applications de règles sont nécessaires pour obtenir a à partir de E_n.

Question 34 –

```
let rec simplifie (expr: exprat): exprat = match expr with
  | Vide -> Vide
  | Epsilon -> Epsilon
  | Lettre c -> Lettre c
  | Union (e1, e2) -> su (Union (simplifie e1, simplifie e2))
  | Concat (e1, e2) -> sc (Concat (simplifie e1, simplifie e2))
  | Etoile e -> se (Etoile (simplifie e)) ;;
```

Question 35 –

```
1  (* On ne vérifie pas la compatibilité des tailles.
2   * On suppose que n > 0 *)
3  let somme (a: mat) (b: mat): mat =
4    let n = Array.length a in
5    let p = Array.length a.(0) in
6    let c = Array.make_matrix n p Vide in
7    for i = 0 to n-1 do
8      for j = 0 to p-1 do
9        c.(i).(j) <- Union (a.(i).(j), b.(i).(j))
10     done;
11   done;
12   c;;
```

Pour la complexité :

→ L'appel à la fonction `Array.make_matrix` s'exécute en temps $\mathcal{O}(np)$.

→ Les boucles `for` parcourent les coordonnées des matrices. Ainsi, la ligne 9 est exécutée np fois et chaque exécution se fait en temps constant.

→ Toutes les autres opérations s'exécutent en temps constant.

En conclusion :

La complexité de `somme` est en $\mathcal{O}(np)$.

Question 36 –

```
1  (* On ne vérifie pas la compatibilité des tailles.
2   * On suppose que n > 0 et p > 0 *)
3  let produit (a: mat) (b: mat): mat =
4    let n = Array.length a in
5    let p = Array.length a.(0) in
6    let q = Array.length b.(0) in
7    let c = Array.make_matrix n q Vide in
8    for i = 0 to n-1 do
9      for j = 0 to q-1 do
10       c.(i).(j) <- Concat (a.(i).(0), b.(0).(j));
11       for k = 1 to p-1 do
12         c.(i).(j) <- Union (c.(i).(j), Concat (a.(i).(k), b.(k).(j)));
13       done;
14     done;
15   done;
16   c;;
```

Pour la complexité :

→ L'appel à la fonction `Array.make_matrix` s'exécute en temps $\mathcal{O}(nq)$.

→ La ligne 10 est exécutée nq fois et s'exécute en temps constant.

→ La ligne 12 est exécutée npq fois et s'exécute en temps constant.

→ Toutes les autres opérations s'exécutent en temps constant.

En conclusion :

La complexité de `produit` est en $\mathcal{O}(npq)$.

Question 37 – Pour $L_{0,0}$, on remarque que tout chemin γ de 0 à 0 dans le graphe de la figure 3 peut être décomposé en une concaténation de $k \in \mathbb{N}$ chemins γ_1, γ_2, …, γ_k où chaque γ_i peut avoir deux formes :

→ Un chemin de taille 1 consistant à emprunter une fois l'arête a. Dans ce cas, γ_i est étiqueté par a.

→ Un chemin empruntant l'arête b, puis $m \in \mathbb{N}$ fois l'arête d, puis l'arête c. Dans ce cas, γ_i est étiqueté par un élément de $\mathscr{L}(bd^\star c)$.

Finalement :

$$\boxed{L_{0,0} = \mathscr{L}\Big((a + bd^\star c)^\star\Big)}$$

Pour $L_{0,1}$, on remarque que tout chemin γ de 0 à 1 dans le graphe de la figure 3 peut être décomposé en une concaténation de deux chemins γ_1, γ_2 où :

→ γ_1 est un chemin de 0 à 0. Ainsi, γ_1 est étiqueté par un élément de $L_{0,0}$.

→ γ_2 un chemin empruntant l'arête b, puis $m \in \mathbb{N}$ fois l'arête d. Ainsi, γ_2 est étiqueté par un élément de $\mathscr{L}(bd^\star)$.

Finalement :

$$\boxed{L_{0,1} = \mathscr{L}\Big((a + bd^\star c)^\star bd^\star\Big)}$$

Pour les deux derniers langages, on reprend les raisonnements ci-dessus en échangeant l'état 0 avec 1, la lettre a avec d et la lettre b avec c :

$$\boxed{L_{1,1} = \mathscr{L}\Big((d + ca^\star b)^\star\Big)} \qquad \boxed{L_{1,0} = \mathscr{L}\Big((d + ca^\star b)^\star ca^\star\Big)}$$

Question 38 – Avec les notations de l'énoncé :

Matrice	A	B	C	D
Nombre de lignes	1	1	$n-1$	$n-1$
Nombre de colonnes	1	$n-1$	1	$n-1$

En utilisant les résultats des questions 35 et 36, voici les temps d'exécution nécessaire pour obtenir les matrices A' et C' (notez qu'à chaque étape, on suppose que les matrices des étapes précédentes ont déjà été calculées) :

→ Le temps d'exécution pour obtenir $D^\star$ est $C(n-1)$.

→ Le temps d'exécution pour obtenir $(a + BD^\star C)$ est :

$$\mathcal{O}(1) + \mathcal{O}(1 \times (n-1) \times (n-1)) + \mathcal{O}(1 \times (n-1) \times 1) = \mathcal{O}(n^2)$$

→ Le temps d'exécution pour obtenir A' est $\mathcal{O}(1)$.

→ Le temps d'exécution pour obtenir C' est :

$$\mathcal{O}((n-1) \times (n-1) \times 1) + \mathcal{O}((n-1) \times 1 \times 1) = \mathcal{O}(n^2)$$

→ Au total, on obtient :

$$C(n-1) + \mathcal{O}(n^2) + \mathcal{O}(1) + \mathcal{O}(n^2) = C(n-1) + \mathcal{O}(n^2).$$

De même pour B' et D', on obtient un temps d'exécution en $C(n-1) + \mathcal{O}(n^2)$.

Pour calculer l'étoile d'une matrice M de taille n, on doit :

→ Découper M en quatre bloc ce qui se fait en temps $\mathcal{O}(n^2)$.

→ Calculer les matrices A', B', C', D' en temps $2C(n-1) + \mathcal{O}(n^2)$ d'après ce qui précède.

→ Recoller les matrice pour obtenir $M^\star$ en temps $\mathcal{O}(n^2)$.

Au total :

La complexité du calcul de l'étoile d'une matrice de taille n vérifie $C(n) = 2C(n-1) + \mathcal{O}(n^2)$.

Si on note T_n le temps d'exécution sans compter les appels récursifs, on obtient :

$$C(1) = T_1 \qquad \forall n \geqslant 2 : C(n) = 2C(n-1) + T_n \text{ avec } T_n = \mathcal{O}(n^2).$$

Par récurrence immédiate sur n :

$$C(n) = \sum_{k=1}^{n} 2^{n-k} T_k = 2^n \sum_{k=1}^{n} \frac{T_k}{2^k}$$

Or $T_k/2^k = \mathcal{O}(k^2/2^k) = o(1/k^2)$, donc la somme converge quand n tend vers $+\infty$. Ainsi :

$$\boxed{C(n) = \mathcal{O}(2^n).}$$

Question 39 – Sous les hypothèses de cette question, les matrices A, B, C, D sont toutes de tailles $n/2 \times n/2$. En suivant le même raisonnement que dans la question précédente :

→ Le calcul de A' et C' se fait en temps :

$$2C(n/2) + \mathcal{O}(n^3)$$

→ Le calcul de B' et D' se fait en temps :

$$2C(n/2) + \mathcal{O}(n^3)$$

Au total :

La complexité du calcul de l'étoile d'une matrice de taille n vérifie $C(n) = 4C(n/2) + \mathcal{O}(n^3)$.

Si on note T_n le temps d'exécution sans compter les appels récursifs, on obtient :

$$C(1) = T_1 \qquad \forall n \geqslant 2 : C(n) = 4C(n/2) + T_n \text{ avec } T_n = \mathcal{O}(n^3).$$

Utilisons le théorème maître pour résoudre cette récurrence. On pose :

$$\alpha = \log_2(4) = 2 \qquad \beta = 3$$

Comme $\beta > \alpha$, le théorème maître donne :

$$C(n) = \mathcal{O}(n^3)$$

Question 40 – Pour gérer le cas des matrices M de taille n quelconque, on peut découper M par blocs ayant pour tailles :

Matrice	A	B	C	D
Nombre de lignes	$\lfloor n/2 \rfloor$	$\lfloor n/2 \rfloor$	$\lceil n/2 \rceil$	$\lceil n/2 \rceil$
Nombre de colonnes	$\lfloor n/2 \rfloor$	$\lceil n/2 \rceil$	$\lfloor n/2 \rfloor$	$\lceil n/2 \rceil$

On remarque en particulier que pour appliquer l'algorithme récursivement, les matrices A et D doivent être carrées, mais B et C peuvent ne pas l'être. Avec ces tailles :

Calcul de A' et C' : $C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + \mathcal{O}(n^3)$

Calcul de B' et D' : $C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + \mathcal{O}(n^3)$

Temps d'exécution total : $C(n) = 2C(\lfloor n/2 \rfloor) + 2C(\lceil n/2 \rceil) + \mathcal{O}(n^3)$

Le théorème maître s'applique comme dans la question précédente et :

On obtient une complexité en $\mathcal{O}(n^3)$.

Question 41 –

```
(* Suppose que m est une matrice carrée *)
let rec etoile_rec (m: mat): mat =
  match Array.length m with
  | n when n <= 0 -> failwith "etoile_rec: n <= 0"
  | 1 -> [|[|Etoile m.(0).(0)|]|]
  | n -> let a,b,c,d = decouper m (n/2) ((n+1)/2) in
         (***)
         let d_et = etoile_rec d in
         let a_p = etoile_rec (somme a (produit b (produit d_et c))) in
         let c_p = produit d_et (produit c a_p) in
         (***)
         let a_et = etoile_rec a in
         let d_p = etoile_rec (somme d (produit c (produit a_et b))) in
         let b_p = produit a_et (produit b d_p) in
         (***)
         recoller a_p b_p c_p d_p;;
```

```
(* Cette fonction n'est pas demandée par l'énoncé mais permet d'avoir un
 * résultat plus lisible en sortie de la fonction 'etoile' *)
let simplifie_mat (m: mat): mat =
  Array.map (fun t -> Array.map simplifie t) m;;
```

```
let etoile (m: mat): mat =
  simplifie_mat (etoile_rec m);;
```

Question 42 –

Pour tout $i \in [\![0, n-1]\!]$, on pose :

$$x_i = \begin{cases} \varnothing & \text{si } i \notin I \\ \varepsilon & \text{si } i \in I \end{cases} \qquad y_i = \begin{cases} \varnothing & \text{si } i \notin F \\ \varepsilon & \text{si } i \in F \end{cases}$$

En effet en utilisant la propriété admise dans l'énoncé :

$$\mathscr{L}([XM_A^\star Y]_{0,0}) = \mathscr{L}\left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_i [M_A^\star]_{i,j} y_j\right) = \bigcup_{i=0}^{n-1}\bigcup_{j=0}^{n-1} \mathscr{L}(x_i) L_{i,j} \mathscr{L}(y_j)$$

On remarque que pour tout $(i,j) \in [\![0, n-1]\!]^2$:

$$\mathscr{L}(x_i) L_{i,j} \mathscr{L}(y_j) = \begin{cases} \emptyset & \text{si } x_i = \varnothing \text{ ou } y_i = \varnothing \\ L_{i,j} & \text{si } x_i = \varepsilon \text{ et } y_i = \varepsilon \end{cases}$$

Ainsi, à l'aide du résultat de la question 9 :

$$\boxed{\mathscr{L}([XM_A^\star Y]_{0,0}) = \bigcup_{i\in I}\bigcup_{j\in F} L_{i,j} = L_A}$$

Question 43 –

```
1  (* Appeler la fonction 'simplifie' n'est pas demandé par l'énoncé, mais permet
2   * d'obtenir un résultat plus lisible. *)
3  let langage (a: automate): exprat =
4    let mA = Array.make_matrix a.nb a.nb Vide in
5    List.iter
6      (fun (q1,c,q2) -> mA.(q1).(q2) <- Union (mA.(q1).(q2), Lettre c))
7      a.trans;
8    let mA_et = etoile mA in
9    let expr = ref Vide in
10   List.iter
11     (fun i -> List.iter
12                 (fun j -> expr := Union (!expr, mA_et.(i).(j)))
13                 a.final )
14     a.init;
15   simplifie !expr;;
```

Pour la complexité, on remarque que :

$$|T| \leqslant |Q \times \Sigma \times Q| = n^2|\Sigma| = \mathcal{O}(n^2)$$

Donc :

→ L'appel à `Array.make_matrix` se fait en $\mathcal{O}(n^2)$.

→ L'appel à `List.iter` ligne 5 se fait en $\mathcal{O}(n^2)$.

→ L'appel à la fonction `etoile` se fait en $\mathcal{O}(n^3)$ (voir la question 40).

→ L'appel à `List.iter` ligne 10 se fait en $\mathcal{O}(n^2)$.

Ainsi :

La fonction `langage` est de complexité $\mathcal{O}(n^3)$.

Question 44 – Pour $E = (ab+b)^\star(ba)$, comme $\varepsilon \in \mathscr{L}((ab+b)^\star)$:

$$\begin{aligned}
\partial_a(E) &= \partial_a\big((ab+b)^\star\big)\cdot\{ba\} \cup \partial_a(ba) \\
&= \partial_a(ab+b)\cdot\{(ab+b)^\star\}\cdot\{ba\} \cup \partial_a(b)\cdot\{a\} \\
&= \Big(\partial_a(ab)\cup\partial_a(b)\Big)\cdot\{(ab+b)^\star ba\} \cup \emptyset\cdot\{a\} \\
&= \Big(\partial_a(a)\cdot\{b\}\cup\emptyset\Big)\cdot\{(ab+b)^\star ba\} \\
&= \Big(\{\varepsilon\}\cdot\{b\}\Big)\cdot\{(ab+b)^\star ba\} \\
&= \Big\{b(ab+b)^\star ba\Big\}
\end{aligned}$$

et :

$$\begin{aligned}
\partial_b(E) &= \partial_b\big((ab+b)^\star\big)\cdot\{ba\} \cup \partial_b(ba) \\
&= \partial_b(ab+b)\cdot\{(ab+b)^\star\}\cdot\{ba\} \cup \partial_b(b)\cdot\{a\} \\
&= \Big(\partial_b(ab)\cup\partial_b(b)\Big)\cdot\{(ab+b)^\star ba\} \cup \{\varepsilon\}\cdot\{a\} \\
&= \Big(\partial_b(a)\cdot\{b\}\cup\{\varepsilon\}\Big)\cdot\{(ab+b)^\star ba\} \cup \{a\} \\
&= \Big(\emptyset\cdot\{b\}\cup\{\varepsilon\}\Big)\cdot\{(ab+b)^\star ba\} \cup \{a\} \\
&= \Big\{(ab+b)^\star ba; a\Big\}
\end{aligned}$$

Question 45 – On prend :

$Q = \{E; E_1; E_2; E_3\}$ où :			
$E = (ab+b)^\star ba$	$E_1 = b(ab+b)^\star ba = b\cdot E$	$E_2 = a$	$E_3 = \varepsilon$

En effet, d'après la question précédente :

$$\partial_a(E) = \{E_1\} \qquad\qquad \partial_b(E) = \{E; E_2\}$$

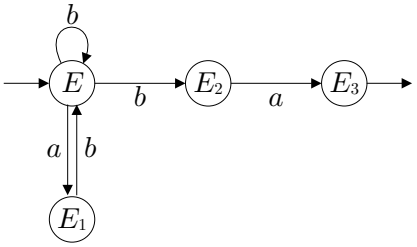
De plus :

$$\partial_a(E_1) = \partial_a(b)\cdot\{E\} = \emptyset\cdot\{E\} = \emptyset \qquad\qquad \partial_b(E_1) = \partial_b(b)\cdot\{E\} = \{\varepsilon\}\cdot\{E\} = \{E\}$$

et :

$$\partial_a(E_2) = \{E_3\} \qquad\qquad \partial_b(E_2) = \partial_a(E_3) = \partial_b(E_3) = \emptyset$$

On obtient le même automate qu'à la question 13 :



Question 46 – Soient u et v deux mots et L un langage, alors :

$$\begin{aligned}
w \in v^{-1}(u^{-1}L) &\Leftrightarrow vw \in u^{-1}L \\
&\Leftrightarrow u(vw) \in L \\
&\Leftrightarrow (uv)w \in L \\
&\Leftrightarrow w \in (uv)^{-1}L
\end{aligned}$$

Question 47 – On montre d'abord que pour tout ensemble de d'expressions rationnelles S, tout mot w et toute lettre x :

$$(\mathcal{P}_1) : w^{-1}\mathscr{L}(S) = \bigcup_{E\in S} w^{-1}\mathscr{L}(E)$$
$$(\mathcal{P}_2) : \mathscr{L}(\partial_w(S)) = \bigcup_{E\in S} \mathscr{L}(\partial_w(E))$$
$$(\mathcal{P}_3) : \mathscr{L}(\partial_x(S)) = x^{-1}\mathscr{L}(S)$$

Pour $(\mathcal{P}_1)$:

$$\begin{aligned} u \in w^{-1}\mathscr{L}(S) &\Leftrightarrow wu \in \mathscr{L}(S) \\ &\Leftrightarrow \exists E \in S : wu \in \mathscr{L}(E) \\ &\Leftrightarrow \exists E \in S : u \in w^{-1}\mathscr{L}(E) \\ &\Leftrightarrow u \in \bigcup_{E\in S} w^{-1}\mathscr{L}(E) \end{aligned}$$

Pour $(\mathcal{P}_2)$:

$$\begin{aligned} w \in \mathscr{L}(\partial_w(S)) &\Leftrightarrow \exists E_0 \in \partial_w(S) : w \in \mathscr{L}(E_0) \\ &\Leftrightarrow \exists E \in S, \exists E_0 \in \partial_w(E) : w \in \mathscr{L}(E_0) \\ &\Leftrightarrow \exists E \in S : w \in \mathscr{L}(\partial_w(E)) \\ &\Leftrightarrow w \in \bigcup_{E\in S} \mathscr{L}(\partial_w(E)) \end{aligned}$$

Pour $(\mathcal{P}_3)$, on utilise $(\mathcal{P}_1)$ et $(\mathcal{P}_2)$:

$$\mathscr{L}(\partial_x(S)) = \bigcup_{E\in S} \mathscr{L}(\partial_x(E)) = \bigcup_{E\in S} x^{-1}\mathscr{L}(E) = x^{-1}\mathscr{L}(S)$$

Prouvons l'égalité demandée dans l'énoncé par récurrence sur $|w|$ en utilisant $(\mathcal{P}_1)$, $(\mathcal{P}_2)$ et $(\mathcal{P}_3)$:

→ Initialisation. Pour $w = \varepsilon$:

$$\mathscr{L}(\partial_\varepsilon(S)) = \bigcup_{E\in S} \mathscr{L}(\partial_\varepsilon(E)) = \bigcup_{E\in S} \mathscr{L}(\{E\}) = \mathscr{L}(S)$$

Et :

$$\varepsilon^{-1}\mathscr{L}(S) = \bigcup_{E\in S} \varepsilon^{-1}\mathscr{L}(E) = \bigcup_{E\in S} \left\{u \in \Sigma^\star : \varepsilon u \in \mathscr{L}(E)\right\} = \bigcup_{E\in S} \mathscr{L}(E) = \mathscr{L}(S)$$

→ Hérédité. Soit $w \in \Sigma^\star$ tel que $\mathscr{L}(\partial_w(S)) = w^{-1}\mathscr{L}(S)$. Pour toute lettre $x \in \Sigma$:

$$\begin{aligned} \mathscr{L}(\partial_{wx}(S)) &= \bigcup_{E\in S} \mathscr{L}(\partial_{wx}(E)) \\ &= \bigcup_{E\in S} \mathscr{L}(\partial_x(\partial_w(E))) \\ &= \bigcup_{E\in S} x^{-1}\mathscr{L}(\partial_w(E)) \\ &= \bigcup_{E\in S} x^{-1}w^{-1}\mathscr{L}(E) \\ &= \bigcup_{E\in S} (wx)^{-1}\mathscr{L}(E) \\ &= (wx)^{-1}\mathscr{L}(S) \end{aligned}$$

Question 48 – Pour tout mot $w \in \Sigma^\star$, on note S_w l'ensemble des états accessibles depuis E en lisant le mot w. Il s'agit de montrer que $\partial_w(E) = S_w$, montrons le par récurrence sur $|w|$:

→ Initialisation. Si $w = \varepsilon$, alors $\partial_\varepsilon(E) = \{E\} = S_\varepsilon$.

→ Hérédité. Soit $w \in \Sigma^\star$ tel que $\partial_w(E) = S_w$. Pour toute lettre $a \in \Sigma$:

$$\begin{aligned} E_1 \in S_{wa} &\Leftrightarrow \exists E_2 \in S_w : (E_2, a, E_1) \in T \\ &\Leftrightarrow \exists E_2 \in S_w : E_1 \in \partial_a(E_2) \\ &\Leftrightarrow E_1 \in \partial_a(S_w) \end{aligned}$$

Ainsi :

$$\partial_{wa}(E) = \partial_a(\partial_w(E)) = \partial_a(S_w) = S_{wa}$$

Question 49 – Pour commencer, montrons que :

$$F \cap \partial_w(E) \neq \emptyset \Leftrightarrow \varepsilon \in \mathscr{L}(\partial_w(E))$$

($\Rightarrow$) S'il existe une expression rationnelle $E_1 \in F \cap \partial_w(E)$, alors $\varepsilon \in \mathscr{L}(E_1)$ par définition de F et donc $\varepsilon \in \mathscr{L}(\partial_w(E))$.

($\Leftarrow$) Si $\varepsilon \in \mathscr{L}(\partial_w(E))$, alors il existe $E_1 \in \partial_w(E)$ tel que $\varepsilon \in \mathscr{L}(E_1)$. Par définition de Q et F : $E_1 \in Q$ et $E_1 \in F$. On obtient bien $F \cap \partial_w(E) \neq \emptyset$

On peut maintenant montrer que si L est le langage reconnu par l'automate d'Antimirov, alors $L = \mathscr{L}(E)$. Pour tout mot $w \in \Sigma^\star$:

$w \in L \Leftrightarrow F \cap \partial_w(E) \neq \emptyset$	d'après la question 48
$\Leftrightarrow \varepsilon \in \mathscr{L}(\partial_w(E))$	d'après l'équivalence montrée ci-dessus
$\Leftrightarrow \varepsilon \in w^{-1}\mathscr{L}(E)$	d'après la question 47
$\Leftrightarrow w\varepsilon \in \mathscr{L}(E)$	
$\Leftrightarrow w \in \mathscr{L}(E)$	

Question 50 – Montrons le par induction sur E :

→ Si $E = \varnothing$ ou $E = \varepsilon$, alors $Q(E) = \emptyset$ et $\|E\| = 0$. D'où le résultat.

→ Si $E = a$ avec $a \in \Sigma$ une lettre, alors $Q(E) = \{\epsilon\}$ et $\|E\| = 1$. D'où le résultat.

→ On suppose le résultat vrai pour deux expressions rationnelles E_1 et E_2. Si on pose $E = E_1 + E_2$, alors $\|E\| = \|E_1\| + \|E_2\|$. De plus, en utilisant la relation (III.1) de l'énoncé :

$$Q(E) = Q(E_0) \cup Q(E_1)$$

Ainsi : $|Q(E)| \leqslant |Q(E_1)| + |Q(E_2)| \leqslant \|E_1\| + \|E_2\| = \|E\|$.

→ On suppose le résultat vrai pour deux expressions rationnelles E_1 et E_2. Si on pose $E = E_1E_2$, alors $\|E\| = \|E_1\| + \|E_2\|$. De plus, en utilisant la relation (III.2) de l'énoncé :

$$Q(E) \subset Q(E_1) \cdot \{E_2\} \cup Q(E_2)$$

Ainsi : $|Q(E)| \leqslant |Q(E_1) \cdot \{E_2\}| + |Q(E_2)| = |Q(E_1)| + |Q(E_2)| \leqslant \|E_1\| + \|E_2\| = \|E\|$.

→ On suppose le résultat vrai pour une expression rationnelle E_1. Si on pose $E = E_1^\star$, alors $\|E\| = \|E_1\|$. De plus, en utilisant la relation (III.3) de l'énoncé :

$$Q(E) \subset Q(E_1) \cdot \{E_1^\star\}$$

Ainsi : $|Q(E)| \leqslant |Q(E_1) \cdot \{E_1^\star\}| = |Q(E_1)| \leqslant \|E_1\| = \|E\|$.

Étant donné que $\delta_\varepsilon(E) = \{E\}$, on en déduit que :

L'automate d'Antimirov contient au plus $\|E\| + 1$ états.