DM5 (éléments de réponses)

Question 1. Une expression régulière du langage reconnu par l'automate donné est $0|0^*1$.

Question 2.

```
let etatsQ = [ 0; 1; 2; 3; 4 ]
let etatsI = [ 0 ]
let etatsF = [ 4 ]
let delta  = [ (0,-1,1) ; (1,-1,2) ; (2,0,1) ; (2,1,4) ; (0,0,3) ; (3,-1,4) ]
```

Opérations sur la structure d'ensemble

Question 3.
□ **3.1.** On peut définir la fonction `cardinal` par :

```
let rec cardinal = function
  | [] -> 0
  | t::r -> 1 + cardinal r
```

□ **3.2.** Estimer sa complexité en fonction de la taille de son ensemble argument. Le nombre d'appels récursifs dans l'appel `cardinal e` est égal au cardinal de `e`.

Question 4.
□ **4.1.** On peut définir la fonction `appartient` par :

```
let rec appartient v = function
  | [] -> false
  | t::r -> v=t || appartient v r
```

□ **4.2.** Le nombre maximal d'appels récursifs est égal à `1+|e|` et que ce nombre est atteint quand `v` n'est pas dans `e`.

Question 5. On définit `ajout` par :

```
let ajout v e =
  if appartient v e then e else v::e
```

Cette fonction ajoute bien `v` à `e` s'il ne s'y trouve pas déjà.

Question 6.
□ **6.1.** Définition de `egalite` :

```
let egalite e1 e2 =
  let rec aux = function
    | [] -> true
    | t::r -> appartient t e2 && aux r
  in cardinal e1 = cardinal e2 && aux e1
```

□ **6.2.** La fonction récursive `aux` teste si un ensemble est inclus dans `e2`, ce qui se vérifie immédiatement par récurrence : `t::r` est inclus dans `e2` si et seulement si `t` appartient à `e2` et `r` est inclus dans `e2`.

Ensuite `e1` est égal à `e2` si et seulement si `e1` est inclus dans `e2` et si `e1` et `e2` ont même nombre d'éléments.

L'appel `appartient t e2` nécessite au plus `|e2|` appels récursifs ; dans le pire cas (quand `e1=e2`) cet appel est effectué pour tous les éléments de `e1` soit une complexité totale majorée par : `|e1|.|e2|`, le test du nombre d'éléments étant lui de complexité `|e1|+|e2|`.

Question 7. La fonction `union` est définie par :

```
let rec union e1 = function
  | [] -> e1
  | t::r -> if appartient t e1 then union e1 r
            else t :: union e1 r
```

Elle parcourt récursivement `e2` et si ses éléments ne sont pas déjà dans `e1` elle les ajoute dans l'union.

Question 8. La fonction `intersection` est définie par :

```
let rec intersection e1 = function
  | [] -> []
  | t::r -> if appartient t e1 then t :: intersection e1 r
            else intersection e1 r
```

Cette fonction parcourt récursivement `e2` et n'en garde que les éléments qui sont déjà dans `e1` ce qui produit bien l'intersection des deux ensembles.

Exploitation des relations de transition

Question 9.

□ **9.1.**

$$\mathrm{s}(\{0\},\delta)=\{1,3\} \qquad \mathrm{s}(\{2\},\delta)=\{1,4\} \qquad \mathrm{s}(\{4\},\delta)=\varnothing \qquad \mathrm{s}(\{1,3\},\delta)=\{2,4\}$$

□ **9.2.**

```
let rec suivants e delta =
  let rec aux t = function
    | [] -> []
    | (q,_,q')::s -> if q = t then q' :: aux t s else aux t s
  in match 0 with
    | [] -> []
    | t::r -> union (aux t delta) (suivants r delta)
```

Question 10.

□ **10.1.**

$$\mathrm{a}(\{0\},\delta)=\{0,1,2,3,4\} \qquad \mathrm{a}(\{2\},\delta)=\{1,2,4\} \qquad \mathrm{a}(\{4\},\delta)=\{1,2,3,4\} \qquad \mathrm{a}(\{1,3\},\delta)=\{4\}$$

□ **10.2.**

▷ **10.2.1.** D'après la définition de A_i, on a bien $A_i \subset A_{i+1}$, ce qui montre que la suite A_i est croissante.

▷ **10.2.2.** Les A_i étant inclus dans l'ensemble fini Q, la suite A_i ne peut être strictement croissante ; il existe donc un $k \in \mathbb{N}$ tel que $A_k = A_{k+1}$.

▷ **10.2.3.** Pour alléger les notations, δ étant fixée, on notera $\mathrm{a}(A)$ au lieu de $\mathrm{a}(A,\delta)$ et $\mathrm{s}(A)$ au lieu de $\mathrm{s}(A,\delta)$.

Soit $d \in \mathrm{a}(\mathrm{s}(A_i))$. Par définition de a, il existe $o' \in \mathrm{s}(A_i)$ et $m \in X^*$ tels que $(o',m,d) \in \delta^*$; il existe ensuite (par définition de s) $o \in A_i$ et $e \in X$ tels que $(o,e,o') \in \delta$; on en déduit que $(o,em,d) \in \delta^*$, et donc que $d \in \mathrm{a}(A_i)$, ce qui prouve que $\mathrm{a}(\mathrm{s}(A_i)) \subset \mathrm{a}(A_i)$.

D'autre part, par définition de a, pour deux sous-ensembles E_1 et E_2 de Q, on a $\mathrm{a}(E_1 \cup E_2) = \mathrm{a}(E_1) \cup \mathrm{a}(E_2)$. On déduit alors : $\mathrm{a}(A_{i+1}) = \mathrm{a}(A_i) \cup \mathrm{a}(\mathrm{s}(A_i))$ et d'après l'inclusion prouvée précédemment : $\mathrm{a}(A_{i+1}) = \mathrm{a}(A_i)$.

▷ **10.2.4.** Soit $E \subset Q$ tel que $\mathrm{s}(E) \subset E$ et $d \in \mathrm{a}(E)$; il existe $o \in E$ et $m \in \Sigma^*$ tels que $(o,m,d) \in \delta^*$. Montrons par récurrence sur la longueur de m que $d \in E$. Si m est le mot vide, par définition de s on a $d \in \mathrm{s}(E)$ et donc $d \in E$ Sinon soit $m = em'$ avec $e \in X$ et $m' \in \Sigma^*$, alors, par définition de δ^*, il existe $q \in Q$ tel que $(o,e,q) \in \delta$ et $(q,m',d) \in \delta^*$. D'où $q \in \mathrm{s}(E) \subset E$ et donc par récurrence $d \in E$. D'où $\mathrm{a}(E) \subseteq E$.

▷ **10.2.5.** D'après les résultats précédents, comme $\mathrm{s}(A_k) \subset A_{k+1} = A_k$ on a $\mathrm{a}(A_k) \subset A_k$; comme par définition de a, on a l'inclusion inverse : $\mathrm{a}(A_k) = A_k$. On a également $\mathrm{a}(E) = \mathrm{a}(A_0) = \mathrm{a}(A_k)$, d'où $\mathrm{a}(E) = A_k$.

□ **10.3.**

```
let access e delta =
  let rec aux e' =
    let e1 = union e' (suivants e' delta) in
      if egalite e' e1 then e' else aux e1
  in aux e
```

□ **10.4.**

```
let rec prefixe q x e = match e with
  | [] -> []
  | t::r -> (q,x,t) :: prefixe q x r
```

Automate fini semi-indéterministe

Question 11.

□ **11.1.** TODO

□ **11.2.**

```
let rec decompose = function
  | [] -> [], []
  | ((_, e, _) as t)::r -> let r1, r2 = decompose r in
      if e = epsilon then t::r1, r2 else r1, t::r2
```

Cette fonction parcourt la liste et trie les transitions suivant qu'elles sont arbitraires ou non.

Question 12.

□ **12.1.** $\bar{\delta}_\varepsilon$ contient, en plus de δ, les transitions (q,ε,q) pour $q \in \{A,B,C,D,E\}$ et la transition (A,ε,C).

□ **12.2.** Si $(q,\varepsilon,q') \in \bar{\delta}_\varepsilon$ (avec $q \neq q'$) il existe une suite finie q_i d'états tels que $q_0 = q$, $q_n = q'$ et pour tout $i = 1,\dots,n$, $(q_{i-1},\varepsilon,q_i) \in \delta_\varepsilon$. On peut supposer (en enlevant d'éventuelles boucles) que les états q_i sont distincts. Une telle chaîne

de n transitions va produire $\binom{n+1}{2}$ éléments dans $\bar{\delta}_\varepsilon$. On peut donc en déduire qu'un majorant de la taille de $\bar{\delta}_\varepsilon$ est $\binom{|\delta_\varepsilon|+1}{2} + |\delta_\varepsilon| + 1$ (en comptant les transitions d'un état sur lui-même).

□ **12.3.** Si l'on identifie (comme le suggère l'énoncé) ε et Λ, $\bar{\delta}_\varepsilon$ est égale à $(\delta_\varepsilon)^*$ (en prenant $X = \varnothing$) ; d'après la définition de a on a $\mathrm{a}(\{o\}) = \{d \in Q \mid (o, \varepsilon, d) \in \delta_\varepsilon^*\}$ (le seul mot possible étant le mot vide) ; ceci entraîne évidemment : $\bar{\delta}_\varepsilon = \{(o, \varepsilon, d) \mid o \in Q, d \in \mathrm{a}(\{o\}, \delta_\varepsilon)\}$.

□ **12.4.**

```
let rec fermeture delta = match delta with
  | [] -> []
  | (q,_,_)::r -> union (prefixe q epsilon (access [q] delta)) (fermeture r)
```

Question 13.

□ **13.1.** Chaque transition de $\delta_\Sigma \triangleright \delta_\varepsilon$ provenant de la composée d'une transition de δ_Σ et d'une de δ_ε, la taille de $\delta_\Sigma \triangleright \delta_\varepsilon$ est majorée par le produit des tailles de δ_Σ et de δ_ε.

□ **13.2.** Quand c'est possible, on prolonge chaque transition de δ_Σ par une transition de $\bar{\delta}_\varepsilon$, et on obtient : $\delta_\Sigma \triangleright \delta_\varepsilon = \{(2,0,1), (2,0,2), (2,1,4), (0,0,3), (0,0,4)\}$.

□ **13.3.**

```
let rec compose delta1 delta2 =
  let rec aux (o,e,q) = function
    | [] -> []
    | (q',_,d)::r -> if q'=q then (o,e,d)::(aux (o,e,q) r)
                             else aux (o,e,q) r
  in match delta1 with
    | [] -> []
    | (o,e,q)::r -> union (aux (o,e,q) delta2) (compose r delta2)
```

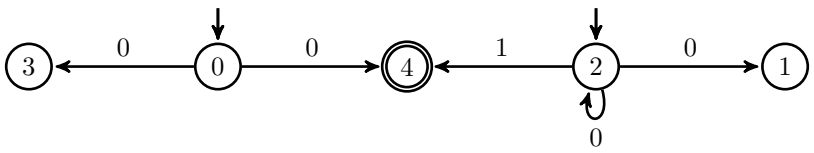
L'appel `aux (o,e,d) g` calcule les transitions obtenues en composant (o, e, d) avec une des transitions de la relation arbitraire g, ce qui se vérifie facilement par récurrence. Puis compose applique aux successivement à tous les éléments de δ_1 et prend l'union du tout.

Question 14.

□ **14.1.** Soit un mot u reconnu par l'automate $\mathcal{A}$. Si u est le mot vide c'est qu'il existe un chemin dans $\mathcal{A}$ étiqueté uniquement par ε qui va d'un état initial i à un état terminal t, ce qui veut dire que $t \in \mathrm{a}(I, \bar{\delta}_\varepsilon)$ donc que t est aussi état initial de $\mathcal{E}(\mathcal{A})$ d'où $\mathcal{E}(\mathcal{A})$ reconnaît aussi le mot vide. Si le mot n'est pas vide : $u = u_1 \dots u_n$, il existe une suite d'états q_i tels que : $q_0 \in I$, $q_n \in T$ et, pour tout $i = 1, \dots, n$, il existe q_i' tel que $(q_{i-1}, u_i, q_i') \in \delta_\Sigma$ et $(q_i', \varepsilon, q_i) \in \bar{\delta}_\varepsilon)$, ce qui entraine que $(q_{i-1}, u_i, q_i) \in \delta_\Sigma \triangleright \bar{\delta}_\varepsilon$, donc que le mot u est reconnu par $\mathcal{E}(\mathcal{A})$. La réciproque se fait de la même façon. Les deux automates $\mathcal{A}$ et $\mathcal{E}(\mathcal{A})$ reconnaisssent donc le même langage.

□ **14.2.** En utilisant les questions précédentes et en remarquant que $\delta_\Sigma \subset \delta_\Sigma \triangleright \bar{\delta}_\varepsilon$, on a comme majorant de la taille de $\delta_\Sigma \triangleright \bar{\delta}_\varepsilon$: $|\delta_\Sigma| . \left(\binom{|\delta_\varepsilon|+1}{2} + |\delta_\varepsilon| + 1\right)$. La déterminisation d'un automate de taille n pouvant donner un automate de taille 2^n, la taille obtenue ici est meilleure.

□ **14.3.**



□ **14.4.** La fonction `semi` traduit seulement la définition de $\mathcal{E}(A)$ en utilisant les fonctions des questions précédentes.

```
let semi (Q, I, T, delta) =
  let deltaX, deltaE = decompose delta in
  let deltaF = fermeture deltaE in
  let delta' = compose deltaX  deltaF in
  let I' = union I (suivants I deltaF)
  in (Q, I', T, delta')
```