DS 2

CORRIGÉ

I Programmation de l'enseignant

▶ **Question 1** L'automate étant supposé complet, il n'y a vraiment aucune difficulté.

```
let rec delta_star auto q word =
  match word with
  | [] -> q
  | letter :: word' -> delta_star auto auto.delta.(q).(letter) word'
```

▶ **Question 2** Si $\mathcal{L}(A)$ est non vide, alors il existe $q_f \in F$ accessible depuis l'état initial q_0. Un chemin de longueur minimal de q_0 à q_f est nécessairement élémentaire, et donc de longueur inférieure ou égale à $|Q| - 1 = n - 1$. L'étiquette de ce chemin fournit un mot de $\mathcal{L}(A)$ de longueur strictement inférieure à n.

▶ **Question 3** On effectue un parcours en largeur à partir de l'état initial jusqu'à atteindre un état final (ou terminer le parcours sans en rencontrer, auquel cas le langage est vide). On utilise une file dont les éléments sont des couples (q, w), où q est un état et w un mot tel que $q_I.\overline{w} = q$ (avec $\overline{w}$ le miroir du mot w).

```
let shortest_word auto =
  let queue = Queue.create () in
  let seen = Array.make auto.nb_states false in
  Queue.push (auto.q0, []) queue;
  seen.(auto.q0) <- true;
  let rec loop () =
    if Queue.is_empty queue then None
    else (
      let (q, word) = Queue.pop queue in
      if auto.accepting.(q) then Some (List.rev word)
      else (
        for letter = 0 to auto.nb_letters - 1 do
          let q' = auto.delta.(q).(letter) in
          if not seen.(q') then (
            seen.(q') <- true;
            Queue.push (q', letter :: word) queue
          )
        done;
        loop ()
      )
    ) in
  loop ()
```

▶ **Question 4** Chaque opération effectuée sur la file est en temps constant, et la création du tableau `seen` en temps $O(|Q|)$. Un état est ajouté au plus une fois dans la file, et l'on enlève un état de la file à chaque passage dans `loop` : on passe donc au plus $|Q|$ fois. Or chaque passage s'effectue en temps $O(|\Sigma|)$: on obtient donc une complexité en $O(|Q| + |Q| \cdot |\Sigma|) = O(|Q| \cdot |\Sigma|)$.

▶ **Question 5** On construit un automate produit (déterministe complet) B comme suit :

- ensemble d'états $Q \times Q'$;
- état initial (q_I, q'_I) ;
- fonction de transition $(q, q').a = (q.a, q'.a)$ (toujours bien défini puisque A et A' sont déterministes complets) ;
- états acceptants $F_B = \{(q, q') \in Q \times Q' \mid q \in F \text{ ou exclusif } q' \in F'\}$.

▶ **Question 6** En posant $n_1 = |Q_1|$ et $n_2 = |Q_2|$, on numérote les états de l'automate produit par $f(i, j) = i \cdot n_2 + j$.

```
let symetric_difference a1 a2 =
  let n1 = a1.nb_states in
  let n2 = a2.nb_states in
  let n = n1 * n2 in
  let m = a1.nb_letters in
  let f q1 q2 = q1 * n2 + q2 in
  let delta = Array.make_matrix n m (-1) in
  for q1 = 0 to n1 - 1 do
    for q2 = 0 to n2 - 1 do
      let q = f q1 q2 in
      for letter = 0 to m - 1 do
        let q' = f a1.delta.(q1).(letter) a2.delta.(q2).(letter) in
        delta.(q).(letter) <- q'
      done
    done
  done;
  let accepting = Array.make n false in
  let xor x y = (x && not y) || (not x && y) in
  for q1 = 0 to n1 - 1 do
    for q2 = 0 to n2 - 1 do
      accepting.(f q1 q2) <- xor a1.accepting.(q1) a2.accepting.(q2)
    done
  done;
  {delta; accepting; q0 = f a1.q0 a2.q0; nb_states = n; nb_letters = m}
```

▶ **Question 7** On a les étapes principales suivantes :

- création de `delta` en $O(n_1 n_2)$;
- calcul de `delta` en $O(n_1 n_2 \cdot |\Sigma|)$;
- création de `accepting` en $O(n_1 n_2)$;
- calcul de `accepting` en $O(n_1 n_2)$.

On obtient une complexité temporelles en $O(|Q_1| \cdot |Q_2| \cdot |\Sigma|)$.

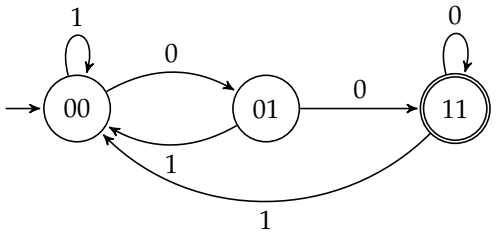
▶ **Question 8**

```
let create_teacher (auto : dfa) : teacher =
  let member w =
    auto.accepting.(delta_star auto auto.q0 w) in
  let counter_example auto' =
    shortest_word (symetric_difference auto auto') in
  {
    member;
    nb_letters = auto.nb_letters;
    counter_example;
  }
```

II Table d'observation

► **Question 9** On a $\mathrm{ligne}(01) \neq \mathrm{ligne}(s)$ pour tout $s \in S$, donc la table n'est pas close. De plus, $\mathrm{ligne}(\varepsilon) = \mathrm{ligne}(0)$ mais $\mathrm{ligne}(1) \neq \mathrm{ligne}(01)$, donc la table n'est pas cohérente.

► **Question 10** On obtient l'automate suivant :



Le langage reconnu est dénoté par $(0|1)^*00$. On vérifie que la table est compatible avec ce langage :

- on a $f(u) = 1$ pour $u \in \{00, 000, 100, 0000\}$;
- on a $f(u) = 0$ pour $u \in \{\varepsilon, 0, 1, 10, 01, 010, 11, 110, 111, 1110\}$.

On a donc bien $f(u) = 1 \Leftrightarrow u \in \mathcal{L}(A)$: l'automate et la table sont compatibles.

► **Question 11**

- Q est bien défini.
- q_0 est bien défini, puisque S clos par préfixe et donc $\varepsilon \in S$.
- si $\mathrm{ligne}(s) = \mathrm{ligne}(s')$ avec $s, s' \in S$, alors :
  - $f(s)$ et $f(s')$ sont bien définis puisque $\varepsilon \in E$ (qui est clos par suffixe) et égaux puisque $\mathrm{ligne}(s) = \mathrm{ligne}(s')$, donc F est bien défini ;
  - pour toute lettre $a \in \Sigma$, $\mathrm{ligne}(sa) = \mathrm{ligne}(s'a)$ puisque la table est cohérente, et il existe $s'' \in S$ tel que $\mathrm{ligne}(sa) = \mathrm{ligne}(s'')$ puisque la table est complète. Donc δ est bien définie (et l'automate est complet et déterministe).

► **Question 12** Par récurrence sur $|s|$:

- si $s = \varepsilon$, alors $\delta^*(q_0, \varepsilon) = q_0$ par définition, et on a bien $q_0 = \mathrm{ligne}(\varepsilon)$;
- sinon, $s = ua$ et $\delta^*(q_0, ua) = \delta(\delta^*(q_0, u), a)$. On a nécessairement $u \in S$ (évident si $s \in S\Sigma$, par fermeture par préfixe de S si $s \in S$), on peut donc appliquer l'hypothèse de récurrence :

$$\begin{aligned} \delta^*(q_0, s) &= \delta(\delta^*(q_0, u), a) \\ &= \delta(\mathrm{ligne}(u), a) \\ &= \delta(\mathrm{ligne}(ua)) && \text{par définition de } \delta \end{aligned}$$

On a donc bien $\delta^*(q_0, s) = \mathrm{ligne}(s)$ pour tout $s \in S \cup S\Sigma$.

► **Question 13** Notons L_A le langage reconnu par $M(S, E, f)$. Notons H_n : « pour tout $s \in S \cup S\Sigma$ et pour tout $e \in E$ tel que $|e| \leqslant n$, on a $f(se) = 1 \Leftrightarrow se \in L_A$ ».

- Dans tous les cas, comme la table est close, on peut considérer $s' \in S$ tel que $\mathrm{ligne}(s') = \mathrm{ligne}(s)$. On a alors par définition $f(se) = f(s'e)$.
- Pour $n = 0$, on a $e = \varepsilon$ et

$$\begin{aligned} se \in L_A &\Leftrightarrow s \in L_A \\ &\Leftrightarrow \delta^*(q_0, s) \in F \\ &\Leftrightarrow \mathrm{ligne}(s) \in F && \text{d'après la question précédente} \\ &\Leftrightarrow \mathrm{ligne}(s') \in F \\ &\Leftrightarrow f(s') = 1 \\ &\Leftrightarrow f(s'e) = 1 \\ &\Leftrightarrow f(se) = 1 \end{aligned}$$

- sinon, $e = ae'$ avec $a \in \Sigma$.

$$\begin{aligned}
se \in L_A &\Leftrightarrow \delta^*(q_0, sae') \in F \\
&\Leftrightarrow \delta^*(\delta^*(q_0, s), ae') \in F \\
&\Leftrightarrow \delta^*(ligne(s), ae') \in F && \text{d'après la question précédente} \\
&\Leftrightarrow \delta^*(ligne(s'), ae') \in F \\
&\Leftrightarrow \delta^*(ligne(s'a), e') \in F && \text{par définition de } \delta \\
&\Leftrightarrow \delta^*(\delta^*(q_0, s'a), e') \in F && \text{d'après la question précédente, avec } s'a \in S\Sigma \\
&\Leftrightarrow \delta^*(q_0, s'ae') \in F && \text{par définition de } \delta^* \\
&\Leftrightarrow s'ae' \in L_A
\end{aligned}$$

  Comme $s' \in S$, on a $s'a \in S \cup S\Sigma$ et comme E est clos par suffixe, on a $e' \in E$ (et $|e'| < |e|$) : on peut donc appliquer l'hypothèse de récurrence à $s'a$ et e'. On obtient donc :

$$\begin{aligned}
se \in L_A &\Leftrightarrow s'ae' \in L_A \\
&\Leftrightarrow f(s'ae') = 1 && \text{par hypothèse de récurrence} \\
&\Leftrightarrow f(s'e) = 1 \\
&\Leftrightarrow f(se) = 1
\end{aligned}$$

On a prouvé que $\forall s \in S \cup S\Sigma,\ \forall e \in E,\ se \in L_A \Leftrightarrow f(se) = 1$: la table et l'automate sont compatibles.

► **Question 14** Soient $n = |Q|$ et $s_1, \dots, s_n \in S$ tels que les $ligne(s_i)$ soient deux à deux distincts, et $q'_i = \delta'^*(q'_0, s_i)$ pour $1 \leqslant i \leqslant n$. Pour $i \neq j$, il existe $e \in E$ tel que $f(s_ie) \neq f(s_je)$ (puisque $ligne(s_i) \neq ligne(s_j)$). Comme A' est compatible avec (S, E, f), on a alors $\delta'^*(q'_0, s_ie) \in F'$ et $\delta'^*(q'_0, s_je) \notin F'$ (ou inversement), et donc $\delta'^*(q'_0, s_ie) \neq \delta'^*(q'_0, s_je)$. On en déduit $q'_i \neq q'_j$: on a donc au moins n états distincts dans Q'. Donc $|Q'| \geqslant |Q|$.

► **Question 15** Pour commencer, on remarque qu'on a nécessairement $|Q| = |Q'|$ d'après la question précédente. Supposons $\delta'^*(q'_0, s) = \delta'^*(q'_0, s')$ pour $s, s' \in S$. On a alors $se \in \mathcal{L}(A') \Leftrightarrow s'e \in \mathcal{L}(A')$ pour tout mot e, et en particulier pour tout $e \in E$. Comme A' est compatible avec (S, E, f), cela implique que $f(se) = f(s'e)$ pour tout $e \in E$, et donc que $ligne(s) = ligne(s')$. Par conséquent, l'application

$$\begin{array}{rccc} \varphi : & Q & \to & Q' \\ & ligne(s) & \mapsto & \delta'^*(q'_0, s) \end{array}$$

est bien définie (l'image ne dépend pas du choix du représentant). D'après la question précédente, elle est injective, et donc bijective par égalité des cardinaux. Il reste à montrer qu'il s'agit d'un isomorphisme.

- Pour l'état initial, $\varphi(q_0) = \varphi(ligne(\varepsilon)) = \delta'^*(q'_0, \varepsilon) = q'_0$.
- Pour les états acceptants :

$$\begin{aligned}
ligne(s) \in F &\Leftrightarrow s \in \mathcal{L}(A) && \text{par définition de } F \\
&\Leftrightarrow f(s) = 1 && \text{puisque } A \text{ est compatible avec } (S, E, f) \\
&\Leftrightarrow s \in \mathcal{L}(A') && \text{puisque } A' \text{ est compatible avec} (S, E, f) \\
&\Leftrightarrow \delta'^*(q'_0, s) \in F' \\
&\Leftrightarrow \varphi(ligne(s)) \in F'
\end{aligned}$$

  Donc $\varphi(F) = F'$.
- Pour les transitions, soit $s \in S$ et $a \in \Sigma$. Il existe $s' \in S$ tel que $ligne(sa) = ligne(s')$ (la table est close), et :

$$\begin{aligned}
\varphi(\delta(ligne(s), a)) &= \varphi(ligne(sa)) && \text{par définition de } \delta \\
&= \varphi(ligne(s'))
\end{aligned}$$

  D'autre part :

$$\begin{aligned}
\delta'(\varphi(ligne(s)), a) &= \delta'(\delta'^*(q'_0, s), a) \\
&= \delta'^*(q'_0, sa)
\end{aligned}$$

Par surjectivité de φ, il existe $s'' \in S$ tel que $q'_0.sa = \varphi(ligne(s'')) = q'_0.s''$. Pour $e \in E$, on a $q'_0.sae = q'_0.s''ae$, donc $f(s''e) = 1 \Leftrightarrow f(sae) = 1$ (puisque A' est compatible avec la table), c'est-à-dire $ligne(s'') = ligne(sa) = ligne(s')$. Finalement, $q'_0.sa = \varphi(ligne(s'')) = \varphi(ligne(s'))$, ce qui permet de conclure.

III Algorithme $\mathcal{L}^*$

► **Question 16**

- La ligne 7 n'est exécutée que si la table n'est pas close, ce qui garantit l'existence de $s \in S$ et $a \in \Sigma$ tel que $ligne(sa) \notin \{ligne(s') \mid s' \in S\}$.
- De même, la ligne 11 n'est exécutée que si la table n'est pas cohérente. Dans ce cas, il existe $s, s' \in S$ et $a \in \Sigma$ tels que $ligne(s) = ligne(s')$ et $ligne(sa) \neq ligne(s'a)$, ce qui signifie qu'il existe $e \in E$ tel que $f(sae) \neq f(s'ae)$.
- Montrons que « S est clos par préfixe » est un invariant.
  - — On initialise S à $\{\varepsilon\}$ qui est clos par préfixe.
  - — Quand on exécute la ligne 8, on ajoute sa à S. Ses préfixes sont :
    - — s, qui est dans S par définition ;
    - — les préfixes propres de s, qui sont dans S d'après l'invariant.

    L'invariant est donc conservé.
  - — Quand on exécute la ligne 17, on ajoute tous les préfixes de w en même temps que w, donc l'invariant est conservé.

  Donc S reste clos par préfixe.
- De même pour E : le seul ajout est ae avec $e \in E$, donc tous les suffixes propres de ae sont déjà dans E.

► **Question 17**

Ligne 3 On teste pour ε et pour les $a \in \Sigma$.

Ligne 9 On teste pour les sae avec $e \in E$.

Ligne 13 On teste pour les sae avec $s \in S \cup S\Sigma$.

Ligne 17 On teste pour les se avec s préfixe de w et $e \in E$.

► **Question 18**

- Si l'on passe dans la première branche, $|(S, E, f)|$ augmente de un puisqu'on ajoute sa à S et que $ligne(sa)$ n'était égal à aucun $ligne(s)$ avec $s \in S$.
- Si l'on passe dans la deuxième branche, $ligne(s)$ est séparé en deux (puisque $ligne(s) \neq ligne(s')$ après l'ajout de ae à E). Donc $|(S, E, f)|$ augment au moins de un (potentiellement plus, d'autres lignes pouvant être séparées).

Donc $|(S, E, f)|$ augmente strictement à chaque passage.

► **Question 19** Considérons $A = M(S, E, f)$ l'automate de la conjecture infructueuse et $A' = M(S', E', f')$ l'automate de la conjecture suivante. A' est compatible avec (S, E, f) (puisque (S', E', f') ne fait qu'étendre cette table), et n'est pas équivalent à A (puisque $w \in \mathcal{L}(A) \oplus \mathcal{L}(A')$). Il ne peut donc pas être isomorphe à A, donc d'après la question **??** il a strictement plus d'états que A. Ce nombre d'états est exactement le nombre de lignes distinctes, donc $|(S, E, f)|$ croît strictement entre deux conjectures successives.

► **Question 20** Soit $A = (\Sigma, Q, q_I, F, \delta)$ déterministe complet compatible avec (S, E, f), et $s, s' \in S$ tels que $ligne(s) \neq ligne(s')$. Il existe donc $e \in E$ tel que $f(se) \neq f(se')$. Comme A est compatible avec (S, E, f), cela signifie que $se \in \mathcal{L}(A)$ et $s'e \notin \mathcal{L}(A)$ (ou inversement). On a donc $q_I.se \neq q_I.s'e$ (où q_I est l'état initial de A), et donc $q_I.s \neq q_I.s'$. On a donc $|Q| \geqslant |(S, E, f)|$.

▶ **Question 21** Remarquons déjà que, si l'algorithme termine, alors il renvoie nécessairement un automate qui reconnaît L.

A_m est compatible avec (S, E, f) à tout moment de l'exécution, donc d'après la question 20 on a $|(S, E, f)| \leqslant n$ tout au long de l'algorithme. Comme de plus $|(S, E, f)|$ croît strictement à chaque passage dans la boucle interne et dans la boucle externe d'après les questions 18 et 19, $n - |(S, E, f)|$ est un variant (entier, positif, strictement décroissant). Donc l'algorithme termine.

L'automate renvoyé reconnaît L : il a donc au moins n états par minimalité de A_m. D'après ce qui précède, il a en fait exactement n états, et la question 15 montre qu'il est isomorphe à A_m.

Remarque

De manière générale, et comme nous l'avons vu en cours, l'automate (déterministe et complet) minimal pour un langage donné est unique à isomorphisme près : il est donc normal que l'automate renvoyé soit isomorphe à A_m.

▶ **Question 22** D'après ce qui précède, l'automate $M(S, E, f)$ de la conjecture possède un nombre $p \leqslant n$ d'états. L'automate produit construit pour la différence symétrique possède donc au plus n^2 états, et reconnaît un langage non vide (puisque la conjecture est incorrecte). L'enseignant renvoie un mot de longueur minimale dans la différence symétrique, donc d'après la question 2 on a $|w| \leqslant n^2$.

▶ **Question 23**

- Initialement, $|S| = |E| = 1$.
- À chaque passage dans la boucle interne, soit $|E|$ soit $|S|$ augmente de un, et $|(S, E, f)|$ augmente au moins de un.
- À chaque conjecture incorrecte, $|S|$ augmente de au plus n^2 d'après la question précédente, $|E|$ reste inchangé et $|(S, E, f)|$ augmente au moins de un.
- Comme $|(S, E, f)|$ est majoré par n, le nombre de conjectures plus le nombre de passages dans la boucle interne est majoré par n.

On en déduit que $|S| \leqslant n^3$ et $|E| \leqslant n$.

▶ **Question 24** En combinant la question 17 et la question précédente, on voit que :

- à chaque passage ligne 9, on fait au plus $|E| \leqslant n$ requêtes ;
- à chaque passage ligne 13, on fait au plus $|S \cup S\Sigma| \leqslant n^3 + |\Sigma|n^3$ requêtes ;
- à chaque passage ligne 18, on faut au plus $|E| \leqslant n$ requêtes pour chaque ua avec u préfixe de w et $a \in \Sigma$. Il y a au plus n^2 préfixes, donc $n^2|\Sigma|$ mots ua et au plus $n^3|\Sigma|$ requêtes au total.

Le nombre total de passages dans toutes ces lignes étant majoré par n, le nombre de requêtes d'appartenance est majoré par $n^4|\Sigma|$.

▶ **Question 25** Les mots de S et de E sont tous de longueur $O(n^2)$: en effet, les mots ajoutés ligne 17 sont de longueur $\leqslant n^2$, et chaque passage ligne 8 ou 12 augmente la longueur maximale d'au plus un. Donc chaque requête d'appartenance se fait en temps $O(n^2)$, et le coût total des requêtes d'appartenance est polynomial (en n et Σ).

Chaque conjecture demande la construction d'un automate produit puis un parcours de cet automate, en temps total $O(n^2\Sigma)$: il y a au plus n conjectures, donc à nouveau polynomial en n et Σ.

Pour déterminer si (S, E, f) est close, il suffit, pour chaque $s \in S$ et $a \in \Sigma$, de parcourir tous les $s' \in S$ pour voir si $\texttt{ligne}(s) = \texttt{ligne}(s')$: si l'on a simplement stocké la matrice codant la table, cela se fait en temps $O(|S|^2 \cdot |\Sigma| \cdot |E|)$, à nouveau polynomial en n et $|\Sigma|$ d'après ce qui précède.

Le calcul de complexité est essentiellement le même pour la cohérence de (S, E, f). Les autres opérations sont des ajouts de ligne et colonne (au pire, on peut recopier toute la table dans une nouvelle table plus grande à chaque fois) et la construction de l'automate, qui se fait clairement en temps polynomial en $|S|$, $|E|$ et Σ, et donc en n et $|\Sigma|$.

Finalement, l'algorithme est bien en temps polynomial en n et $|\Sigma|$, pour n'importe quel choix raisonnable de structure de données pour la table.

▶ **Question 26** On suit exactement la définition de l'énoncé.

```
let construct_auto table =
  let n = Obs.nb_rows table in
  let m = Obs.nb_letters table in

  (* Détermination des états acceptants *)
  let accepting = Array.make n false in
  let process_word word =
    let i = Obs.get_row_number table word in
    accepting.(i) <- Obs.compute_f table word [] in
  Obs.iter_s table process_word;

  (* Calcul des transitions *)
  let delta = Array.make_matrix n m (-1) in
  let add_transitions word =
    let q = Obs.get_row_number table word in
    for letter = 0 to m - 1 do
      let q' = Obs.get_row_number table (word @ [letter]) in
      delta.(q).(letter) <- q'
    done in
  Obs.iter_s table add_transitions;

  (* État initial *)
  let q0 = Obs.get_row_number table [] in

  {accepting; q0; delta; nb_states = n; nb_letters = m}
```

▶ **Question 27** On itère sur tous les mots de S pour déterminer les numéros de ligne i tels qu'aucun $s \in S$ ne vérifie $numero(ligne(s)) = i$: c'est le rôle du tableau `present`. D'autre part, on remplit le tableau `preimage` avec des éléments de $S\Sigma$ de manière à avoir $numero(ligne(preimage[i])) = i$ pour tout i. On parcourt ensuite le tableau `present` :

- si toutes les cases valent `true`, la table est complète ;
- sinon, on trouve un numéro de ligne i qui ne correspond à aucun mot de S, et l'on lève l'exception `Incomplete u` avec $u \in S\Sigma$ tel que $numero(ligne(u)) = i$.

```
let check_complete table =
  let n = Obs.nb_rows table in
  let present = Array.make n false in
  let preimage = Array.make n [] in
  let process_s_word s =
    present.(Obs.get_row_number table s) <- true in
  Obs.iter_s table process_s_word;
  let process_sa_word sa =
    preimage.(Obs.get_row_number table sa) <- sa in
  Obs.iter_sa table process_sa_word;
  for i = 0 to n - 1 do
    if not present.(i) then raise (Incomplete preimage.(i))
  done
```

▶ **Question 28** On définit une fonction `check : word -> word -> unit` qui prend en entrée deux mots s et s' et :

- ne fait rien si $ligne(sa) = ligne(s'a)$ pour toute lettre $a \in \Sigma$;
- sinon, lève l'exception `Inconsistent w`, avec $w \in E$ tel que $ligne(saw) \neq ligne(s'aw)$.

Il faut ensuite appeler cette fonction sur toutes les paires $\{s, s'\}$ d'éléments de S telles que $\text{ligne}(s) = \text{ligne}(s')$. Pour ce faire :

- on construit un tableau `preimages` tel que `preimages.(i)` contiennent la liste des mots $s \in S$ tels que $\text{ligne}(s) = i$;
- on appelle `check_class` sur chacune de ces listes, ce qui a pour effet d'appeler `check` sur toutes les paires $\{s, s'\}$ où s et s' sont dans la liste.

```
let check_consistent (table : Obs.t) =
  let rec check s s' =
    for letter = 0 to Obs.nb_letters table - 1 do
      match Obs.separate_rows table (s @ [letter]) (s' @ [letter]) with
      | None -> ()
      | Some word -> raise (Inconsistent (letter :: word))
    done in
  let preimages = Array.make (Obs.nb_rows table) [] in
  let process_word s =
    let i = Obs.get_row_number table s in
    preimages.(i) <- s :: preimages.(i) in
  Obs.iter_s table process_word;
  let rec check_class = function
    | s :: rest -> List.iter (check s) rest; check_class rest
    | _ -> () in
  Array.iter check_class preimages
```

► **Question 29** C'est très simple si l'on a compris la manière dont les deux fonctions précédentes ont été conçues ; essayer d'utiliser une boucle **while**, en revanche, risque de rendre le code très compliqué.

```
let rec make_complete_and_coherent table teacher =
  try
    check_complete table;
    check_consistent table
  with
  | Incomplete word ->
    Obs.add_to_s table word teacher.member;
    make_complete_and_coherent table teacher
  | Inconsistent word ->
    Obs.add_to_e table word teacher.member;
    make_complete_and_coherent table teacher
```

► **Question 30** Tout le travail a déjà été fait.

```
let learn teacher =
  let table = Obs.initial_table teacher.nb_letters teacher.member in
  let rec outer_loop () =
    make_complete_and_coherent table teacher;
    let auto = construct_auto table in
    match teacher.counter_example auto with
    | None -> auto
    | Some w ->
      Obs.add_to_s table w teacher.member;
      outer_loop () in
  outer_loop ()
```