DS* 3

APPRENTISSAGE D'UN LANGAGE RÉGULIER

Durée : 4 heures

On s'intéresse dans ce sujet à la possibilité pour un *élève* d'apprendre un langage régulier en interagissant avec un *enseignant*. L'apprentissage est ici « inductif » : l'enseignant ne transmet pas de règle à l'élève, mais est capable de répondre (correctement) à des questions posées par l'élève.

Plus précisément, on considère un langage régulier L sur un alphabet Σ. L est connu de l'enseignant mais pas de l'élève, Σ est en revanche connu des deux. Un enseignant est dit *minimalement compétent* s'il répond correctement aux deux types suivants de questions :

- les *requêtes d'appartenance*, où l'élève fournit un mot w et l'enseignant indique si $w \in L$;
- les *conjectures*, où l'élève soumet une description d'un langage régulier X, et où l'enseignant :
  - répond *Correct* si $X = L$;
  - fournit sinon un élément de la différence symétrique de X et L, élément que l'on appelle *contre-exemple*.

  La description de X fournie par l'élève peut *a priori* être une expression régulière ou un automate fini, déterministe ou pas. Dans ce qui suit, l'élève fera en pratique ses conjectures sous la forme d'un automate fini déterministe complet A tel que $\mathcal{L}(A) = X$.

Remarque

On rappelle que la différence symétrique $L \oplus X$ de L et X est l'ensemble $(L \setminus X) \cup (X \setminus L)$ des mots qui appartiennent à L mais pas à X, ou inversement. On a $L \oplus X = \emptyset$ si et seulement si $L = X$.

Le but du problème est l'étude d'un algorithme, appelé $\mathcal{L}^\star$, qui permet à l'élève de déterminer exactement le langage L avec une complexité raisonnable (autant en nombre de requêtes à l'enseignant qu'en temps de calcul pour l'élève).

Indications pour la programmation Toutes les questions de programmation sont à traiter en langage OCaml. Les fonctions des modules `Array` et `List` peuvent être librement utilisées (mais il est conseillé de rappeler la signature et la spécification d'éventuelles fonctions « exotiques » que vous décideriez d'utiliser), ainsi que les fonctions définies dans le module ouvert pas défaut (comme `min`, `max`, `@`...). Certaines fonctions du module `Queue` seront utiles : elles sont rappelées en annexe à la fin du sujet.

I Programmation de l'enseignant

Dans tout le sujet, on suppose que l'alphabet Σ est de la forme $[0 \dots n-1]$ pour un certain n. Une lettre de Σ sera donc un entier, et un mot une liste de lettres :

```
type letter = int
type word = letter list
```

Pour représenter un automate fini déterministe complet, on utilise le type suivant :

```
type dfa =
  {q0 : int;
   nb_letters : int;
   nb_states : int;
   accepting : bool array;
   delta : int array array}
```

- L'alphabet Σ est $[0 \dots \texttt{nb\_letters} - 1]$.

- L'ensemble Q d'états est $[0 \dots \texttt{nb\_states} - 1]$.
- `q0` indique l'état initial (et appartient donc à $[0 \dots \texttt{nb\_states} - 1]$) ;
- Pour $0 \leqslant q < \texttt{nb\_states}$, `accepting.(q)` vaut `true` si l'état q est acceptant, `false` sinon.
- `delta` est un tableau de longueur `nb_states`, et pour $0 \leqslant q < \texttt{nb\_states}$, `delta.(q)` est un tableau de longueur `nb_letters` tel que, pour $0 \leqslant x < \texttt{nb\_letters}$, `delta.(q).(x)` indique l'état $\delta(q, x)$.

Remarques

- Tous les automates considérés dans le sujet seront supposés déterministes et complets.
- Les champs `nb_letters` et `nb_states` sont redondants (on pourrait calculer leur valeur à partir des dimensions du tableau `delta`) ; ils sont inclus pour clarifier le code.

▶ **Question 1** Écrire une fonction `delta_star` telle que l'appel `delta_star auto q w` renvoie l'état $\delta^*(q, w)$, et indiquer sa complexité.

```
val delta_star : dfa -> int -> word -> int
```

▶ **Question 2** Soit A un automate à n états. Montrer que si $\mathcal{L}(A)$ est non vide, alors il contient un mot de longueur strictement inférieure à n.

▶ **Question 3** Écrire une fonction `shortest_word` qui prend en entrée un automate A et renvoyant :

- `Some w`, où `w` est un mot de longueur minimale de $\mathcal{L}(A)$, s'il en existe un :
- `None` sinon (c'est-à-dire si $\mathcal{L}(A)$ est vide).

```
val shortest_word : dfa -> word option
```

▶ **Question 4** Indiquer, en la justifiant rapidement, la complexité de la fonction `shortest_word`, en fonction de $n = |Q|$ et $p = |\Sigma|$.

▶ **Question 5** Étant donnés deux automates A et A' sur un même alphabet Σ, indiquer comment construire un automate B reconnaissant le langage $\mathcal{L}(A) \oplus \mathcal{L}(A')$ (différence symétrique des deux langages).

▶ **Question 6** Écrire une fonction `symetric_difference` prenant en entrée deux automates A et A' sur un même alphabet, et renvoyant l'automate B de la question précédente.

```
val symetric_difference : dfa -> dfa -> dfa
```

▶ **Question 7** Déterminer la complexité de la fonction `symetric_difference`.

On définit le type suivant pour représenter un enseignant minimalement compétent associé à un langage L :

```
type teacher = {
  nb_letters : int;
  member : word -> bool;
  counter_example : dfa -> word option;
}
```

- `nb_letters` spécifie l'alphabet $\Sigma = [0 \dots \texttt{nb\_letters} - 1]$.
- `member` prend en entrée un mot de Σ^* et renvoie un booléen indiquant s'il appartient au langage L.
- `counter_example auto` renvoie :
  - `None` si le langage reconnu par `auto` est égal à L ;
  - `Some w`, où w est un contre-exemple (élément de $L \oplus \mathcal{L}(\texttt{auto})$), sinon.

▸ **Question 8** Écrire une fonction `create_teacher` prenant en entrée un automate déterministe A tel que $\mathcal{L}(A) = L$ et renvoyant un enseignant adapté. L'enseignant renvoyé fournira systématiquement des contre-exemples de longueur minimale (un tel enseignant sera dit *raisonnablement compétent*).

```
val create_teacher : dfa -> teacher
```

II Table d'observation

L'algorithme utilisé par l'élève va effectuer des requêtes d'appartenance dans un ordre bien défini, et organiser les résultats de ces requêtes dans une *table d'observation*, que l'on définit ci-dessous.

- Un ensemble $X \subseteq \Sigma^*$ est dit *clos par préfixe* s'il vérifie la propriété suivante : si $w \in X$, alors tous les préfixes de w sont dans X.
- De même, X est *clos par suffixe* s'il contient tous les suffixes de w dès qu'il contient w.
- On remarquera que si X est non vide et clos par préfixe, alors $\varepsilon \in X$ (et de même si X est clos par suffixe).
- Une *table d'observation* est un triplet (S, E, f) tel que :
  - $S \subset \Sigma^*$ est un ensemble non vide et clos par préfixe (le « S » signifie *Start*, cet ensemble contient des débuts de mots) ;
  - $E \subset \Sigma^*$ est un ensemble non vide et clos par suffixe (le « E » signifie *End*, cet ensemble contient des fins de mots) ;
  - $f : (S \cup S\Sigma)E \to \{0, 1\}$.
- Une table d'observation et un langage X sont dits *compatibles* si, pour tout couple $s, e \in (S \cup S\Sigma) \times E$, on a :
$$f(se) = 1 \Leftrightarrow se \in X.$$
- Une table d'observation et un automate A sont dits compatibles si la table est compatible avec $\mathcal{L}(A)$.
- À une application f, on peut associer une matrice T dont les lignes sont indexées par $S \cup S\Sigma$ et les colonnes indexées par E. Pour un mot $s \in S \cup S\Sigma$, l'application partielle $e \mapsto f(se)$ correspond alors à une « ligne » de cette matrice, et on la note $\mathtt{ligne}(s)$. Autrement dit :
$$\begin{array}{rccc} \mathtt{ligne}(s) : & E & \to & \{0,1\} \\ & e & \mapsto & f(se) \end{array}$$

Un exemple La matrice T_0 ci-dessous spécifie la fonction f d'une table d'observation (S, E, f) avec $S = \{\varepsilon, 0, 1, 10\}$ et $E = \{10, 0, \varepsilon\}$.

		ε	0	10
S	ε	0	0	1
	0	0	0	1
	1	1	1	0
	10	1	0	1
$S\Sigma \setminus S$	00	0	0	1
	01	0	0	0
	11	0	0	1
	100	0	0	0
	101	1	1	0

Figure 1 – La table T_0.

On a ici $(S \cup S\Sigma)E = \{\varepsilon, 0, 1, 00, 10, 010, 100, 110, 1010\}$. La première ligne indique que $f(\varepsilon) = 0$, $f(0) = 0$ et $f(10) = 1$. La quatrième ligne indique que $f(10) = 1$, $f(100) = 0$ et $f(1010) = 1$.

La matrice T_0 contient 24 coefficients, alors que $|(S \cup S\Sigma)| = 9$: on a donc des informations redondantes. Par exemple, l'image de 10 est donnée à la fois dans la première ligne ($\varepsilon \cdot 10$), dans la troisième ligne ($1 \cdot 0$) et dans la quatrième ligne ($10 \cdot \varepsilon$).

Définition

- Une table d'observation est dite *cohérente* si, pour tous $s, s' \in S$ tels que $\mathrm{ligne}(s) = \mathrm{ligne}(s')$ et pour tout $a \in \Sigma$, on a $\mathrm{ligne}(sa) = \mathrm{ligne}(s'a)$.
- Elle est dite *close* si, pour tout $t \in S\Sigma$, il existe $s \in S$ tel que $\mathrm{ligne}(t) = \mathrm{ligne}(s)$.

► **Question 9** Montrer que la table de la figure 1 n'est ni close ni cohérente.

Dans toute la suite de cette partie, on suppose que (S, E, T) est une table d'observation close et cohérente, et l'on définit l'automate $M(S, E, f) = (\Sigma, Q, q_0, F, \delta)$ par :

- $Q = \{\mathrm{ligne}(s) \mid s \in S\}$;
- $q_0 = \mathrm{ligne}(\varepsilon)$;
- $F = \{\mathrm{ligne}(s) \mid s \in S \text{ et } f(s) = 1\}$;
- $\delta(\mathrm{ligne}(s), a) = \mathrm{ligne}(sa)$ pour $s \in S$ et $a \in \Sigma$.

► **Question 10** Montrer que $M(S, E, f)$ correctement défini, et qu'il s'agit d'un automate déterministe complet.

► **Question 11** On considère la table d'observation close et cohérente ci-dessous.

		ε	0
S	ε	0	0
	0	0	1
	1	0	0
	00	1	1
$S\Sigma \setminus S$	01	0	0
	10	0	1
	11	0	0
	000	1	1
	111	0	0

Figure 2 – La table T_1.

Donner l'automate $A = M(S, E, f)$ associé, et déterminer le langage $\mathcal{L}(A)$ reconnu par cet automate. Justifier que la table est compatible avec A.

► **Question 12** Montrer que pour tout $s \in S \cup S\Sigma$, on a $\delta^*(q_0, s) = \mathrm{ligne}(s)$.

► **Question 13** Montrer que $M(S, E, f)$ est compatible avec (S, E, f).

► **Question 14** Soit $A' = (\Sigma, Q', q'_0, F', \delta')$ compatible avec (S, E, f). Montrer que $|Q'| \geqslant |Q|$.

Définition Deux automates (déterministes et complets) $A_1 = (\Sigma, Q_1, q_0^1, F_1, \delta_1)$ et $A_2 = (\Sigma, Q_2, q_0^2, F_2, \delta_2)$ sont dits *isomorphes* s'il existe une application $\varphi : Q_1 \to Q_2$ telle que :

(1) φ est bijective ;

(2) $\varphi(q_0^1) = q_0^2$;

(3) $\varphi(F_1) = F_2$;

(4) $\varphi(\delta_1(q, a)) = \delta_2(\varphi(q), a)$ pour $q \in Q_1$ et $a \in \Sigma$.

► **Question 15** Montrer que si $A' = (\Sigma, Q', q'_0, F', \delta')$ est un automate compatible avec (S, E, F) tel que $|Q'| \leqslant |Q|$, alors A est isomorphe à $M(S, E, f)$.

III Algorithme $\mathcal{L}^\star$

L'algorithme $\mathcal{L}^\star$ utilisé par l'élève maintient à jour une table d'observation (S, E, f), où f est en fait stockée sous la forme d'une matrice T. À chaque fois qu'il ajoute des mots à S ou à E, l'algorithme effectue des requêtes d'appartenance pour étendre f (en ajoutant des lignes ou des colonnes à la matrice T).

Algorithme 1 – Apprentissage $\mathcal{L}^\star$

```
1:  S ← {ε}
2:  E ← {ε}
3:  Calculer f à l'aide de requêtes d'appartenance.
4:  répéter
5:      tant que (S, E, f) n'est pas close ou pas cohérente faire
6:          si (S, E, f) n'est pas close alors
7:              Trouver s ∈ S et a ∈ Σ tels que ligne(sa) ≠ ligne(s') pour tout s' ∈ S.
8:              S ← S ∪ {sa}
9:              Étendre f à (S ∪ SΣ)E en effectuant des requêtes d'appartenance.
10:         sinon si (S, E, f) n'est pas cohérente alors
11:             Trouver s, s' ∈ S, a ∈ Σ et e ∈ E tels que ligne(s) = ligne(s') et f(sae) ≠ f(s'ae).
12:             E ← E ∪ {ae}
13:             Étendre f à (S ∪ SΣ)E en effectuant des requêtes d'appartenance.
14:     M ← M(S, E, f)
15:     Soumettre à l'enseignant la conjecture M.
16:     si l'enseignant répond par un contre-exemple w alors
17:         Ajouter w et ses préfixes à S.
18:         Étendre f à (S ∪ SΣ)E en effectuant des requêtes d'appartenance.
19:     sinon
20:         renvoyer M
```

► **Question 16** Justifier que les lignes commençant par « Trouver. . .tels que. . . » (lignes 7 et 11) ne peuvent échouer et que S (respectivement E) reste toujours clos par préfixe (respectivement par suffixe).

► **Question 17** Donner l'ensemble des requêtes d'appartenance que l'élève effectue aux lignes 3, 9, 13 et 18.

On définit :

- A_m un automate déterministe complet minimal (en nombre d'états) pour le langage L – on note n son nombre d'états ;
- $|(S, E, f)| \stackrel{\text{déf}}{=} \left|\{\text{ligne}(s) \mid s \in S\}\right|$ le nombre de lignes *distinctes* correspondant à des mots de S dans la matrice T (où (S, E, f) est une table d'observation).

► **Question 18** Montrer que $|(S, E, f)|$ croît strictement à chaque passage dans la boucle interne.

► **Question 19** Montrer que $|(S, E, f)|$ croît strictement entre une conjecture (incorrecte) et la conjecture suivante.

► **Question 20** Soit (S, E, f) une table d'observation, dont on ne suppose ni qu'elle est close ni qu'elle est cohérente. Montrer que si A est un automate (déterministe et complet) compatible avec (S, E, f), alors A possède au moins $|(S, E, f)|$ états.

► **Question 21** Montrer que l'algorithme $\mathcal{L}^\star$ termine et renvoie un automate isomorphe à A_m.

On suppose à présent que l'enseignant utilisé est celui programmé à la partie I, construit à partir de l'automate minimal A_m du langage L.

► **Question 22** Majorer en fonction de n la longueur maximale d'un contre-exemple w donné en réponse à une conjecture incorrecte.

▶ **Question 23** Majorer en fonction de n et $|\Sigma|$ le cardinal de S et de E.

▶ **Question 24** Majorer en fonction de n le nombre de requêtes d'appartenance et de conjectures effectuées par l'algorithme $\mathcal{L}^\star$.

▶ **Question 25** Justifier, en esquissant une réalisation très simple de la structure de table d'observation, que l'algorithme $\mathcal{L}^\star$ peut être implémenté avec une complexité polynomiale en n.

IV Programmation de l'élève

IV.1 Construction de l'automate

On suppose pour l'instant que l'on dispose d'une structure de table d'observation, avec l'interface suivante (toutes les fonctions ne sont pas nécessairement immédiatement utiles) :

```
(* Le type d'une table d'obsevation *)
type t

(* Renvoie le nombre de lignes *distinctes* de la table. *)
(* Les lignes sont numérotées de 0 à nb_rows - 1. *)
val nb_rows : t -> int

(* Renvoie la taille de l'alphabet *)
val nb_letters : t -> int

(* Renvoie le numéro de la ligne correspondant à un mot de S ∪ SΣ. *)
val get_row_number : t -> word -> int

(* Prend en entrée un mot w ∈ S ∪ SΣ et un mot e ∈ E *)
(* et renvoie f(w,e). *)
val compute_f : t -> word -> word -> bool

(* Applique une fonction g successivement à tous les mots de l'ensemble S. *)
val iter_s : t -> (word -> unit) -> unit

(* Applique une fonction g successivement à tous les mots de l'ensemble SΣ. *)
val iter_sa : t -> (word -> unit) -> unit
```

On suppose que ces fonctions sont regroupées dans un module `Obs` : ainsi, on pourra appeler la fonction `nb_rows` par `Obs.nb_rows` et son type sera `Obs.t -> int`.

▶ **Question 26** Écrire une fonction `construct_auto` qui prend en entrée une table d'observation (S, E, f), supposée close et cohérente, et renvoie l'automate $M(S, E, f)$.

```
val construct_auto : Obs.t -> dfa
```

IV.2 Test de complétude et de fermeture

On définit les deux exceptions suivantes :

```
exception Incomplete of word
exception Inconsistent of word
```

▸ **Question 27** Écrire une fonction `check_complete` qui prend en entrée une table et lève l'exception `Incomplete w`, où w est un mot de $S\Sigma$ tel que $\text{ligne}(w) \neq \text{ligne}(u)$ pour tout $u \in S$, si la table n'est pas close.

```
val check_complete : Obs.t -> unit
```

On ajoute à la signature du module `Obs` la fonction suivante :

```
val separate_rows : Obs.t -> word -> word
```

L'appel `Obs.separate_rows table u u'`, dont le comportement n'est défini que si u et u' appartiennent à $S \cup S\Sigma$, renvoie :

- `None` si $\text{ligne}(u) = \text{ligne}(u')$;
- `Some w`, où w est un mot de E tel que $f(uw) \neq f(u'w)$ sinon.

▸ **Question 28** Écrire une fonction `check_consistent` qui prend en entrée une table d'observation et :

- ne fait rien si la table est cohérente ;
- si la table est incohérente, lève l'exception `Inconsistent w`, où w est un mot de la forme aw' tel que :
  - $a \in \Sigma$;
  - $w' \in E$;
  - il existe $u, u' \in S$ tels que $\text{ligne}(u) = \text{ligne}(u')$ et $f(uaw) \neq f(u'aw)$.

```
val check_consistent : Obs.t -> unit
```

IV.3 Algorithme $\mathcal{L}^\star$

On ajoute à la signature du module `Obs` les deux fonctions suivantes :

```
val add_to_s : t -> word -> (word -> bool) -> unit

val add_to_e : t -> word -> (word -> bool) -> unit
```

- Ces deux fonctions prennent en argument une table d'observation `t`, un mot `w` et une fonction `member` permettant d'effectuer une requête d'appartenance au langage L que l'on cherche à apprendre (`member u` renvoie `true` si et seulement si $u \in L$).
- La fonction `add_to_s` ajoute le mot w passé en argument, ainsi que tous ses préfixes, à l'ensemble S. Elle met également à jour la fonction f en ajoutant les lignes nécessaires pour le nouvel ensemble $S \cup S\Sigma$.
- La fonction `add_to_e` ajoute le mot w passé en argument à E. Elle met également à jour la fonction f en ajoutant la nouvelle colonne correspondant à w.

▸ **Question 29** Écrire une fonction `make_complete_and_coherent` prenant en entrée une table et un enseignant, et modifiant la table pour la rendre close et cohérente. Cette fonction doit donc réaliser la boucle interne des lignes 5 à 13 de l'algorithme 1.

```
val make_complete_and_coherent : table -> teacher -> unit
```

On ajoute à la signature du module `Obs` la fonction suivante :

```
val initial_table : teacher -> t
```

Cette fonction renvoie la table d'observation pour $S = E = \{\varepsilon\}$ avec l'enseignant fourni (elle effectue donc l'initialisation des lignes 1 à 3 de l'algorithme 1).

▶ **Question 30** Écrire une fonction `learn` qui prend en entrée un enseignant pour un langage L et renvoie un automate déterministe minimal reconnaissant L, en suivant l'algorithme $\mathcal{L}^\star$.

```
val learn : teacher -> dfa
```

Ce sujet est directement adapté de l'article *Learning Regular Sets from Queries and Counter-Examples*, publié en 1987 par Dana Angluin, professeure à Yale et contributrice majeure au développement de l'apprentissage automatique. Cet article a eu un impact assez important, et de nombreuses variantes de l'algorithme L* ont ensuite été développées.

Vous êtes invités à réfléchir à des implémentations possibles du module **`Obs`**, basées par exemple sur des *tries* pour réaliser des dictionnaires dont l'ensemble des clés est un ensemble clos par préfixe (ou suffixe) de mots.

Annexe

Module `Queue`

Toutes les fonctions ci-dessous ont une complexité en $O(1)$. L'utilisation d'autres fonctions du module est autorisée, à condition de rappeler leur spécification ; cependant, les fonctions fournies suffisent pour traiter le sujet.

```
(* Crée une file vide *)
Queue.create : unit -> 'a Queue.t

(* Test de vacuité *)
Queue.is_empty : 'a Queue.t -> bool

(* Ajout d'un élément *)
Queue.push : 'a -> 'a Queue.t -> unit

(* Extraction de l'élément le plus ancien *)
(* Lève l'exception Queue.Empty si la file est vide *)
Queue.pop : 'a Queue.t -> 'a
```