TD/TP Bonus n°2

Langages réguliers

Dans ce TD, l'objectif est de démontrer des résultats sur la classe des langages réguliers $\mathbf{LR}(\Sigma)$. On n'utilisera pas des résultats du cours sur les inclusions de classes, et sur les propriétés de stabilité, sauf précisé autrement. Le but est de *réviser* avant les écrits/oraux. Certains exercices sont tirés de sujets d'annales, d'autres non.

Dans toute cette fiche d'exercice, on possède un alphabet Σ fini non vide. On notera $\mathbf{Rec}(\Sigma)$ l'ensemble des langages reconnaissables par des automates finis. On notera $\mathrm{Reg}(\Sigma)$ l'ensemble des expressions régulières défini inductivement dans le cours.[1]

Les questions « plus TD » seront notées en rose, et celles « plus TP » seront notées en bleu.

Exercice I. Construction de Thompson (TD/TP OCaml)

Dans cet exercice, on supposera pouvoir supprimer les ε-transitions.

I.1. Définition théorique

Q1. Construire un automate $\mathscr{A}_\emptyset$ reconnaissant le langage $\emptyset$.

Q2. Construire un automate $\mathscr{A}_\varepsilon$ reconnaissant le langage $\{\varepsilon\}$.

Q3. Construire un automate $\mathscr{A}_a$ reconnaissant le langage $\{a\}$, quel que soit $a \in \Sigma$.

Pour les questions suivantes, on pourra procéder par dessin pour donner l'idée générale, mais on précisera également la construction ensembliste du quintuplet $\mathscr{A} = (Q_\mathscr{A}, \Sigma_\mathscr{A}, I_\mathscr{A}, F_\mathscr{A}, \delta_\mathscr{A})$. Ce sera très utile dans la Section I.3.

Q4. Étant donnés deux automates $\mathscr{A}_e$ et $\mathscr{A}_f$ reconnaissant les langages $\mathscr{L}(e)$ et $\mathscr{L}(f)$ respectivement, construire un automate $\mathscr{A}_{e\cdot f}$ reconnaissant $\mathscr{L}(e \cdot f)$.

Q5. Étant donnés deux automates $\mathscr{A}_e$ et $\mathscr{A}_f$ reconnaissant les langages $\mathscr{L}(e)$ et $\mathscr{L}(f)$ respectivement, construire un automate $\mathscr{A}_{e \mid f}$ reconnaissant $\mathscr{L}(e \mid f)$.

Q6. Étant donné un automate $\mathscr{A}_e$ reconnaissant le langage $\mathscr{L}(e)$, construire un automate $\mathscr{A}_{e^\star}$ reconnaissant le langage $\mathscr{L}(e^\star)$.

Q7. En déduire $\mathbf{LR}(\Sigma) \subseteq \mathbf{Rec}(\Sigma)$.

> **Remarque.** Cet exercice donne une construction alternative à la construction de Glushkov (*i.e.* l'algorithme de Berry-Sethi), bien plus visuelle. Mais, cet algorithme génère des ε-transitions qu'il faut supprimer. Comparons ces deux constructions.

I.2. Mieux que Glushkov ?

Soit e l'expression régulière :

[1] Il est noté $\mathscr{E}_{\mathrm{reg}}(\Sigma)$ dans le poly de cours.

$$e = (a(ab)^\star)^\star \mid (ab)^\star$$

Q8. Construire, avec la construction de Glushkov, un automate $\mathscr{A}_{\text{Glushkov}}$ reconnaissant $\mathscr{L}(e)$.

Q9. Construire, avec la construction de Thompson, un automate $\mathscr{A}_{\text{Thompson}}$ reconnaissant $\mathscr{L}(e)$. On appliquera *exactement* cette construction comme décrite dans les questions précédentes.

Q10. Comparer les deux automates : nombre d'états, « complexité » apparente[2], nombre de transitions, *etc.*

I.3. Implémentons cet algorithme !

On représentera un automate en OCaml par :

```
type automaton = {
  states: int list;
  initial_states: int list;
  final_states: int list;
  transitions: (int * char * int) list;
}
```

où le type `'a list` représentera un ensemble d'éléments de type `'a` par une liste. L'ordre des éléments n'a pas d'importance, mais chaque élément de l'ensemble ne doit apparaître qu'une fois. Dans la définition du type `automaton`, l'alphabet Σ n'est pas précisé. En effet, on supposera utiliser les lettres de l'ensemble $\{a, \dots, z, A, \dots, Z\}$. Une ε-transition sera représenté par le 3-uplet $(p, \&, q)$.

I.3.a. Quelques manipulations d'ensembles

Q11. Coder `add : 'a -> 'a list -> 'a list` qui construit l'ensemble $E \cup \{x\}$ lors de l'appel `add` $E\ x$.

Q12. Coder `union : 'a list -> 'a list -> 'a list` qui construit l'ensemble $E \cup F$ lors de l'appel `union` $E\ F$.

On pourra définir l'opérateur `@@` avec `let (@@) = union`.

Q13. Coder `new_val : int list -> int` une fonction qui trouve un entier `i` qui n'est pas dans la liste `l` lors de l'appel `new_val l`.

I.3.b. Construction de l'automate

On définit le type `reg` en OCaml par :

```
type reg =
  | EmptySet
  | EmptyWord
  | Letter of char
  | Star of reg
  | Concatenation of reg * reg
  | Union of reg * reg
```

Q14. Écrire une fonction `thompson : reg -> automaton` en suivant la construction définie précédemment.

[2] Si l'automate a l'air plus complexe visuellement (notion très subjective, mais bon . . .)

Exercice II. Automates produits (TD)

Dans cet exercice, nous allons démontrer des résultats de stabilité de $\mathbf{LR}(\Sigma)$ en utilisant les *automates produits*. On supposera que les automates manipulés sont déterministes et complets ; on prendra soin lors de construction d'automates que cette condition est bien vérifiée.

Q1. Démontrer (en recopiant bêtement le cours) la stabilité de $\mathbf{LR}(\Sigma)$ par intersection.[3]

> **Remarque.** Cette construction est simple d'un point de vue théorique. Mais, d'un point de vue pratique, elle est bien trop couteuse : on génère des ε-transitions qu'il faut supprimer, et l'automate n'est pas forcément déterministe après passage au complémentaire. Dans cet exercice, on donne une méthode construisant un automate déterministe sans ε-transitions : les automates produits.

Soient $\mathscr{A} = (Q_{\mathscr{A}}, \Sigma, I_{\mathscr{A}}, F_{\mathscr{A}}, \delta_{\mathscr{A}})$ et $\mathscr{B} = (Q_{\mathscr{B}}, \Sigma, I_{\mathscr{B}}, F_{\mathscr{B}}, \delta_{\mathscr{B}})$ deux automates finis déterministes complets sur un même alphabet Σ.

On construit l'automate $\mathscr{C} = (Q_{\mathscr{C}}, \Sigma, I_{\mathscr{C}}, F_{\mathscr{C}}, \delta_{\mathscr{C}})$ sur l'alphabet Σ en posant :
- $Q_{\mathscr{C}} = Q_{\mathscr{A}} \times Q_{\mathscr{B}}$ (d'où le nom, automate *produit*) ;
- $I_{\mathscr{C}} = I_{\mathscr{A}} \times I_{\mathscr{B}}$;
- $F_{\mathscr{C}} = \{(q_{\mathscr{A}}, q_{\mathscr{B}}) \in Q_{\mathscr{C}} \mid q_{\mathscr{A}} \in F_{\mathscr{A}} \quad \underline{\text{et}} \quad q_{\mathscr{B}} \in F_{\mathscr{B}}\}$;
- $\delta_{\mathscr{C}} = \left\{(q_{\mathscr{A}}, q_{\mathscr{B}}) \xrightarrow{a} (q'_{\mathscr{A}}, q'_{\mathscr{B}}) \,\middle|\, a \in \Sigma \text{ et } \left(q_{\mathscr{A}} \xrightarrow{a} q'_{\mathscr{A}}\right) \in \delta_{\mathscr{A}} \text{ et } \left(q_{\mathscr{B}} \xrightarrow{a} q'_{\mathscr{B}}\right) \in \delta_{\mathscr{B}}\right\}$.

Q2. Construire deux automates $\mathscr{A}$ et $\mathscr{B}$ reconnaissant respectivement les langages $\mathscr{L}\left((a(ab)^\star)^\star\right)$ et $\mathscr{L}\left((ab)^\star\right)$.[4] Dans cet exemple, exprimer le langage reconnu par $\mathscr{C}$ en fonction de $\mathscr{L}(\mathscr{A})$ et de $\mathscr{L}(\mathscr{B})$.

Q3. Dans le cas général, exprimer le langage reconnu par $\mathscr{C}$ en fonction de $\mathscr{L}(\mathscr{A})$ et de $\mathscr{L}(\mathscr{B})$. Le démontrer par double-inclusion.

Q4. En adaptant la construction de $\mathscr{C}$, démontrer la stabilité de $\mathbf{LR}(\Sigma)$ par :
- intersection,
- union,
- différence symétrique,
- différence ensembliste.

Parmi ces opérations, une correspond à celle donnée en Q2, mais je ne peux pas dire laquelle, cela *spoilerai* l'exercice.

Exercice III. Algorithme de Conway (TD/TP OCaml)

Dans cet exercice, on étudie une alternative à l'algorithme d'élimination d'états. L'idée est de convertir un automate en une matrice, puis de calculer l'*étoile* de cette matrice, pour obtenir une expression régulière.

Dans tout cet exercice, on considère $\mathscr{A} = (Q, \Sigma, I, F, \delta)$. Quitte à numéroter les états, on supposera que $Q = [\![1, n]\!]$, où $n = \operatorname{card} Q$. Avec cet algorithme, on démontrera l'inclusion $\mathbf{Rec}(\Sigma) \subseteq \mathbf{LR}(\Sigma)$.

On considère des matrices qui contiennent des expressions régulières. On notera $\mathscr{M}^{\text{reg}}_{n,m}$ l'ensemble de ces matrices de taille $n \times m$. Pour une matrice $\boldsymbol{A} \in \mathscr{M}^{\text{reg}}_{n,m}$, on notera $A_{i,j}$ ou $[\boldsymbol{A}]_{i,j}$ le coefficient à la i-ème ligne et la j-ème colonne. Pour deux matrices $\boldsymbol{A}$ et $\boldsymbol{B}$ d'expressions régulières, on définit
- $[\boldsymbol{A} + \boldsymbol{B}]_{i,j} = A_{i,j} \mid B_{i,j}$, que l'on étendra avec la notation $\sum$;

[3] Ça devrait ressembler à : $L_1 \cap L_2 = (L_1^{\mathsf{c}} \cup L_2^{\mathsf{c}})^{\mathsf{c}}$, où $L^{\mathsf{c}} = \wp(\Sigma)^\star \setminus L$.
[4] Si vous avez fait l'exercice I, vous avez déjà fait 90% du travail.

- $[\boldsymbol{A} \times \boldsymbol{B}]_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$, que l'on étendra avec la notation $\prod$.
- $\boldsymbol{A}^{k+1} = \boldsymbol{A}^k \cdot \boldsymbol{A}$, et $\boldsymbol{A}^0 = \mathbf{I}_n = \mathbf{diag}(\varepsilon, \dots, \varepsilon)$[5]

Ces définitions correspondent aux matrices usuelles où l'addition correspond à l'union, la multiplication entre scalaires correspond à la concaténation. L'équivalent de 1 est ε, l'équivalent de 0 est $\emptyset$.

On définit en OCaml le type :

```
type matreg = reg array array
```

où `reg` est défini en ici dans l'exercice 1.

III.1. Quelques opérations matricielles

Q1. Coder une fonction `sum : matreg -> matreg -> matreg` calculant la somme de deux matrices. On supposera que les dimensions des matrices correspondent pour la somme.

Q2. Coder une fonction `prod : matreg -> matreg -> matreg` calculant le produit de deux matrices. On supposera que les dimensions des matrices correspondent pour le produit.

Q3. Coder une fonction `pow : matreg -> int -> matreg` calculant la puissance d'une matrice. On supposera la matrice carrée.

III.2. Représentation d'un automate

Dans les questions suivantes, on utilisera le type `automaton` qui correspond à un automate généralisé défini par :

```
type automaton = {
  states: int list;
  initial_states: int list;
  final_states: int list;
  transitions: (int * regexp * int) list;
}
```

Étant donné un automate $\mathscr{A}$, on définit la *matrice* de cet automate comme $\boldsymbol{M}_{\mathscr{A}} \in \mathscr{M}_{n,n}^{\text{reg}}$ par :

$$[\boldsymbol{M}_{\mathscr{A}}]_{p,q} = \sum_{(p,a,q)\in\delta_{\mathscr{A}}} a$$

Si cette somme ne contient aucun élément, on considèrera qu'il s'agit de l'expression régulière $\emptyset$.

> **Remarque.** Cette représentation correspond à la matrice d'adjacence de l'automate, vu comme un graphe. La représentation par un quintuplet correspond plus à une représentation par liste d'adjacence. Les questions ci-dessous permettent de basculer d'une représentation à l'autre.

> **Remarque.** Attention ! Avec le passage de $\mathscr{A}$ à $\boldsymbol{M}_{\mathscr{A}}$, on perd certaines informations. Par exemple, quels états sont initiaux, quels états sont finaux. Dans la suite de l'exercice, cela ne posera pas problème : on ne donnera donc aucun état initial ni état final dans Q5.

Q4. Définir `mat_of_auto : automaton -> matreg` qui calcule $\boldsymbol{M}_{\mathscr{A}}$ pour $\mathscr{A}$ donné.

Q5. Définir `auto_of_mat : matreg -> automaton` qui calcule $\mathscr{A}$ tel que $\boldsymbol{M}_{\mathscr{A}} = \boldsymbol{M}$ pour une matrice $\boldsymbol{M}$ donnée.

[5] Sur la diagonale, on place des ε ; les autres coefficients sont $\emptyset$.

On cherche maintenant à définir l'*étoile de Kleene* sur une matrice $\boldsymbol{A} \in \mathscr{M}_{n,m}^{\text{reg}}$.

III.3. Étoile de Kleene d'une matrice

Commençons par un cas simple, le cas d'une matrice 2×2. Considérons une matrice $\boldsymbol{M}$ définie par :

$$\boldsymbol{M} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

où $\{a, b, c, d\} \subseteq \Sigma$.

Q6. Dessiner l'automate (partiel) représenté par la matrice $\boldsymbol{M}$. Comme indiqué dans la remarque, on ne définira aucun état initial ni final.

On note $\mathscr{A}_{i,j} = (\{0,1\}, \{i\}, \{j\}, \delta)$ où $\delta = \left\{ (p, M_{p,q}, q) \,\middle|\, (p,q) \in \{0,1\}^2 \right\}$. Il correspond à l'automate partiel de la question précédente, où i est l'état initial et j l'état final.

Q7. Donner une expression régulière qui représente chacun des langages $\mathscr{L}(\mathscr{A}_{i,j})$ pour chaque couple $(i,j) \in \{0,1\}^2$.

Définition. En s'inspirant des relations trouvées en Q7, on propose une représentation matricielle de l'étoile de Kleene d'une matrice. On considère une matrice $\boldsymbol{M}$ de taille supérieur à 2 définie par bloc, et on définit la matrice $\boldsymbol{M}^\star$ par bloc comme :

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \quad \rightsquigarrow \quad \boldsymbol{M}^\star = \begin{pmatrix} \boldsymbol{A}' & \boldsymbol{B}' \\ \boldsymbol{C}' & \boldsymbol{D}' \end{pmatrix}$$

où les blocs $\boldsymbol{A}$ et $\boldsymbol{D}$ sont carrés, et les blocs de $\boldsymbol{M}^\star$ sont donnés par :
- $\boldsymbol{A}' = (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}^\star\boldsymbol{C})^\star$,
- $\boldsymbol{B}' = \boldsymbol{A}^\star\boldsymbol{B}(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}^\star\boldsymbol{B})^\star$,
- $\boldsymbol{C}' = \boldsymbol{D}^\star\boldsymbol{C}(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}^\star\boldsymbol{C})^\star$,
- et $\boldsymbol{D}' = (\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}^\star\boldsymbol{B})^\star$.

Dans le cas où la matrice $\boldsymbol{M}$ est de taille égale à 1, alors $\boldsymbol{M} = (e)$ et donc $\boldsymbol{M}^\star = (e^\star)$.

On admet que, pour deux découpages donnés, les coefficients des deux matrices (vus comme des expressions régulières) sont équivalents.

Q8. Vérifier que l'on ait bien $\boldsymbol{M} \cdot \boldsymbol{M}^\star + \mathbf{I}_n \equiv \boldsymbol{M}^\star$.

Remarque. Une question reste en suspens… Quel découpage de $\boldsymbol{M}$ est plus efficace ? C'est ce que nous allons analyser.

On notera, dans les questions suivantes, $C(n)$ la complexité du calcul de $\boldsymbol{M}^\star$, pour $\boldsymbol{M} \in \mathscr{M}_{n,n}^{\text{reg}}$. Attention, elle dépendra du découpage choisi.

Q9. On considère les blocs $\boldsymbol{A}$ et $\boldsymbol{D}$ de taille respectivement 1×1 et $(n-1) \times (n-1)$. Quelle est la complexité des différentes sommes et produits ? Montrer ainsi que $C(n) = 2C(n-1) + \mathrm{O}(n^2)$. En déduire la complexité de cet algorithme.

Q10. On supposera n une puissance de 2. On considère les blocs $\boldsymbol{A}$ et $\boldsymbol{D}$ de taille $(n/2) \times (n/2)$ pour les deux blocs. Quelle est la complexité des différentes sommes et produits ? Montrer également la relation $C(n) = 4C(n/2) + \mathrm{O}(n^3)$. En déduire la complexité de cet algorithme.

Q11. Comment gérer le cas où n est quelconque ? Quelle complexité pour le calcul de $\boldsymbol{M}^\star$?

Indication OCaml. Dans le code mis à disposition, on donne les deux fonctions suivantes :

- `split : mat -> int -> int -> (mat * mat * mat * mat)` qui découpe une matrice $\boldsymbol{M}$ en 4 blocs $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, et $\boldsymbol{D}$ de telle sorte que le bloc $\boldsymbol{A}$ soit de taille $n \times n$ et $\boldsymbol{D}$ soit de taille $m \times m$, où les entiers n et m sont donnés en argument (dans cet ordre).
- `join : mat -> mat -> mat -> mat -> mat` qui renvoie la matrice

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}$$

où les blocs $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ et $\boldsymbol{D}$ sont donnés et on des tailles compatibles.

Q12. Coder une fonction `star : mat -> mat` calculant l'étoile de Kleene d'une matrice. On utilisera l'algorithme adéquat.

III.4. L'algorithme de Conway

On admettra le résultat suivant : quel que soit le couple $(i, j) \in Q^2$, nous avons

$$\mathscr{L}\left([\boldsymbol{M}^\star]_{i,j}\right) = \mathscr{L}(\mathscr{A}_{i,j})$$

où $\mathscr{L}_{i,j}$ est une généralisation de ce qui est défini avant Q7.

Q13. Montrer que $\mathscr{L}(\mathscr{A}) = \mathscr{L}\left([\boldsymbol{X}\boldsymbol{M}_{\mathscr{A}}^\star\boldsymbol{Y}]_{0,0}\right)$, où $\boldsymbol{X} = (x_1 \ \cdots \ x_n)$ et $\boldsymbol{Y} = (y_1 \ \cdots \ y_n)^\mathsf{T}$, avec la contrainte $x_1, \dots, x_n, y_1, \dots, y_n \in \{\emptyset, \varepsilon\}$. On précisera les valeurs de X et Y en fonction de $\mathscr{A}$.

Q14. Écrire une fonction `lang : automaton -> reg` calculant le langage, sous forme d'une expression régulière, de l'automate. Quelle est la complexité de cette fonction ?

Q15. Conclure $\mathbf{Rec}(\Sigma) \subseteq \mathbf{LR}(\Sigma)$.

Exercice IV. Déterminisation d'un automate (TP C)

Le but de cet exercice est de coder l'algorithme de déterminisation d'un automate. Dans tout l'exercice, on supposera l'automate sans ε-transitions.

Q1. Rappeler la construction d'un automate déterministe à partir d'un automate non déterministe.

On appellera, dans la suite, $\det(\mathscr{A})$ l'automate déterministe équivalent calculé avec la construction précédente. Cette construction a un problème[6] : les états de $\det(\mathscr{A})$ sont des ensembles d'états.

Si l'on représente un automate par un type `'a automaton`, où `'a` est le type des états de l'automate, alors la fonction $\det$ aura pour signature : `det : 'a automaton -> ('a set) automaton`. Le type d'un automate non déterministe à son déterminisé *change*. Ceci n'est certes pas un problème dans un langage comme OCaml, mais en C, c'est bien plus complexe.

On supposera que les états de l'automate initial sont des entiers de l'intervalle $[\![0, n-1]\!]$. On cherche donc à encoder un élément de $\wp([\![0, n-1]\!])$ sous la forme d'entiers. Voici la solution que l'on propose :

$$\begin{aligned} \text{numéro} : \wp([\![0, n-1]\!]) &\longrightarrow [\![0, 2^n - 1]\!] \\ I &\longmapsto \sum_{i \in I} 2^i \end{aligned}$$

On vérifie aisément que cette fonction est injective (elle est même bijective).

[6] Par problème, on entend qu'elle peut poser problème dans certains cas, et notamment au moment de l'implémentation.

Q2. Définir en C un type `automaton` :
- pour les ensembles d'états, on utilisera des tableaux d'entiers (on n'oubliera pas de stocker la taille du tableau),
- pour l'alphabet, on utilisera un tableau de caractères,
- pour l'ensemble des transitions, on utilisera un tableau de transitions et on définira une structure transition.

Indication C. Pour nous aider dans cette tâche, on utilise des *opérations bit à bit*. En effet, la fonction numéro consiste à placer des `1` dans la représentation binaire de $\text{numéro}(I)$ au i-ème bit, pour tout $i \in I$, comme montré ci-dessous.

$$I = \{0, \ , 2, 3, \ \}$$
$$\text{numéro}(I) = \ \texttt{1 0 1 1 0}$$

On introduit plusieurs opérateurs sur les entiers. Ces opérateurs réalisent des opérations sur la représentation binaire de des deux nombres considérés. Ils s'utilisent avec des éléments de type `int`. On définit :

(1) le « et » logique, `&` en C, tel que a `&` $b = \overline{\left([a]_{61} \wedge [b]_{61}\right) \cdots \left([a]_0 \wedge [b]_0\right)}$;

(2) le « ou » logique, `|` en C, tel que a `|` $b = \overline{\left([a]_{61} \vee [b]_{61}\right) \cdots \left([a]_0 \vee [b]_0\right)}$;

(3) le « xor » logique,[7] `^` en C, tel que a `^` $b = \overline{\left([a]_{61} \oplus [b]_{61}\right) \cdots \left([a]_0 \oplus [b]_0\right)}$;

(4) le « non » logique, `~` en C, tel que `~` $a = \overline{\left(\neg[a]_{61}\right) \cdots \left(\neg[a]_0\right)}$;

(5) le « décalage vers la gauche », `<<` en C, tel que a `<<` $b = \overline{[a]_{61-b} \cdots [a]_0 \ 0 \cdots 0}$;

(6) le « décalage vers la droite », `>>` en C, tel que a `>>` $b = \overline{0 \cdots 0 \ [a]_{61} \cdots [a]_b}$.

L'exemple considéré dans le tableau ci-dessous (Tableau 1) montre l'effet des opérations définies ci-avant sur les entiers $a = 6$ et $b = 3$.

Expression	Représentation binaire	Résultat numérique
a	`0000···0110`	6
b	`0000···0011`	3
a `&` b	`0000···0010`	2
a `|` b	`0000···0111`	7
a `^` b	`0000···0101`	5
a `<<` b	`0···0110000`	48
a `>>` b	`0000000···0`	0

Tableau 1. Exemple sur les opérations logiques sur les entiers C

L'entier $2^{\texttt{n}}$ peut être obtenu aisément à l'aide de l'expression `1 << n`.

Q3. En utilisant *habilement* l'indication, définir une fonction `bool is_in(int, int)` qui teste si un état q (1er argument) est dans l'ensemble X (2ème argument, représenté par l'entier $\text{numéro}(X)$). Cette fonction aura une complexité en $\mathrm{O}(1)$.

[7] Le « xor » vient de l'anglais *exclusive or* : il s'agit du « ou exclusif ».

Q4. Implémenter la fonction numéro avec une fonction `int numéro(int* tab, int len)` où l'on donne `tab` un tableau d'entiers, et `len` sa longueur. Par exemple, avec le tableau

$$\ell = [1; 5; 2; 5; 2; 5; 2; 2; 1; 2; 1]$$

la fonction retournera $38 = 2^1 + 2^2 + 2^5$, car le tableau ℓ représente l'ensemble $X = \{1, 2, 5\}$.

Définition. Pour un automate partiel $\mathscr{A}$, on peut définir une *fonction de transition*

$$\begin{aligned}\hat{\delta} : Q_{\mathscr{A}} \times \Sigma &\longrightarrow \wp(Q_{\mathscr{A}}) \\ (q, a) &\longmapsto \{p \mid (q, a, p) \in \delta_{\mathscr{A}}\}.\end{aligned}$$

La représentation des transitions de $\mathscr{A}$ par un ensemble de transitions ou par une fonction de transitions sont équivalentes.

Définition. On étend la définition de $\hat{\delta}$ aux ensembles d'états en posant la fonction suivante :

$$\begin{aligned}\hat{\Delta} : \wp(Q_{\mathscr{A}}) \times \Sigma &\longrightarrow \wp(Q_{\mathscr{A}}) \\ (X, a) &\longmapsto \bigcup_{q \in X} \hat{\delta}(q, a).\end{aligned}$$

Q5. Écrire une fonction `int delta(int, char)` qui implémente la fonction $\hat{\Delta}$. Le premier argument est $\text{numéro}(X)$, et le second est `a`. On retournera donc $\text{numéro}\big(\hat{\Delta}(X, a)\big)$.

Q6. Réécrire la réponse de Q1 en utilisant la fonction $\hat{\Delta}$.

Q7. Implémenter la fonction `automaton det(automaton)` qui calcule $\det(\mathscr{A})$ pour $\mathscr{A}$ donné. On n'oubliera pas de commenter le code donné. Cette question est bien plus longue que les autres.

Q8. Quelle est la complexité de la fonction `det` ?

Q9. En pratique, quelle est la limitation majeur de cette implémentation ?[8]

Remarque. En pratique, plusieurs états (de l'automate déterminisé) ne sont pas utilisés. On pourra donc émonder l'automate.[9] L'exercice s'arrête ici, mais je vous invite à implémenter un algorithme d'émondage d'automate.

Exercice V. Automate miroir (TD/TP OCaml)

Dans cet exercice, on considère un automate $\mathscr{A}$, et on s'intéresse à l'automate miroir de l'automate $\mathscr{A}$, noté $\tilde{\mathscr{A}}$ ou $\text{transpose}\ \mathscr{A}$.

Définition. Étant donné $\mathscr{A} = (Q_{\mathscr{A}}, \Sigma, I_{\mathscr{A}}, F_{\mathscr{A}}, \delta_{\mathscr{A}})$ un automate, on définit son *automate miroir* comme l'automate $\tilde{\mathscr{A}} = (Q_{\mathscr{A}}, \Sigma, F_{\mathscr{A}}, I_{\mathscr{A}}, \delta_{\tilde{\mathscr{A}}})$, où

$$\delta_{\tilde{\mathscr{A}}} = \{(p, a, q) \,|(q, a, p) \in \delta_{\mathscr{A}}\}.$$

On définit la fonction $\text{transpose} : \mathscr{A} \mapsto \tilde{\mathscr{A}}$.

Définition. Étant donné un mot $w = w_0 \dots w_n$, le *renversé* de w est $w^{\text{rev}} = w_n \dots w_0$. Étant donné un langage L, le *renversé* de L est

$$L^{\text{rev}} = \{w^{\text{rev}} \mid w \in L\}.$$

[8] *Indice* : regarder les types utilisés dans `automaton`, et leurs limites.
[9] *i.e* supprimer les états qui ne sont pas accessibles et co-accessibles.

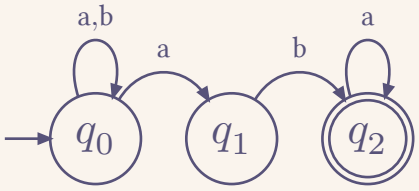

Fig. 1. Un automate exemple $\mathscr{A}_{\text{ex}}$

Q1. Quel est le langage $\mathscr{L}(\mathscr{A}_{\text{ex}})$ de l'automate de la Fig. 1 ? Déterminer $\mathscr{L}(\mathscr{A}_{\text{ex}})^{\text{rev}}$.

Q2. Dessiner $\tilde{\mathscr{A}}$, et exprimer son langage. Que remarquez vous ?

Normalement, vous remarquez quelque chose d'intéressant … Si ce n'est pas le cas, réessayez jusqu'à que vous y arriviez.[10]

Q3. Généraliser l'observation de Q2 quel que soit l'automate $\mathscr{A}$.[11] On démontrera cette observation.

Q4. Implémenter la fonction `transpose : automaton -> automaton`. Quelle est sa complexité ?

Si vous avez fait les exercices 1 et 3, vous pouvez faire la question suivante.

Q5. Coder une fonction `reverse : reg -> reg` qui, à une expression régulière e, associe une expression régulière e' telle que $\mathscr{L}(e') = (\mathscr{L}(e))^{\text{rev}}$.

Exercice VI. Minimisation d'automates (TD/TP OCaml)

Dans cet exercice, on s'intéresse à un problème très intéressant[12] : minimiser des automates.

ATTENTION ! Cet exercice n'est pas complet.

Définition. On dit de deux automates $\mathscr{A}$ et $\mathscr{B}$ qu'ils sont *équivalents* si $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B})$. On le note alors $\mathscr{A} \equiv \mathscr{B}$.

Définition. On dit qu'un automate $\mathscr{A}$ est *minimal* si le nombre d'état qu'il possède est minimal pour le langage de l'automate :

$$\operatorname{card} Q_{\mathscr{A}} = \min_{\mathscr{A}' \equiv \mathscr{A}} \operatorname{card} Q_{\mathscr{A}'}.$$

On étudiera deux méthodes de minimisation d'automates dans les deux parties suivants. Ces parties sont indépendantes.

Q1. Justifier que la notion d'automate minimal est bien définie.

VI.1. Algorithme de Brzozowsky

L'implémentation de cet algorithme est bien plus simple (avec quelques fonctions usuelles pour la manipulation d'automates) que celle de la partie suivante. Pour montrer cela, voici l'algorithme *complet* dans la Fig. 2.

[10] *Indice* : les questions Q1 et Q2 sont liées.
[11] *Indice* : Démontrer que $\mathscr{L}(\tilde{\mathscr{A}}) = (\mathscr{L}(\mathscr{A}))^{\text{rev}}$.
[12] Tous les problèmes de cette fiche sont très intéressants, mais celui l'est bien plus

Entrée. Un automate $\mathscr{A}$

Sortie. Un automate minimal $\mathscr{B}$

$\mathscr{R} \leftarrow$ émonde(det(transpose $\mathscr{A}$))
$\mathscr{B} \leftarrow$ émonde(det(transpose $\mathscr{R}$))
Retourner $\mathscr{B}$

Fig. 2. Algorithme de Brzozowsky

Simple, non ? Les questions suivantes vont s'intéresser aux différentes parties de cet algorithme.

Q2. Si cela n'est pas déjà fait, faire l'exercice 5. Il concerne transpose $\mathscr{A}$, et les propriétés de l'automate miroir.

Définition. Étant donné un automate déterministe complet $\mathscr{A}$, on définit la *fonction de transition étendue*, notée $\delta^\star$ ou $\delta^\star_{\mathscr{A}}$ en cas d'ambigüité, comme
- $\delta^\star(q, \varepsilon) = q$
- $\delta^\star(q, a \cdot w) = \delta^\star(p, w)$, où p est l'unique état vérifiant $(q, a, p) \in \delta_{\mathscr{A}}$

Définition. Pour un langage $L \subseteq \Sigma^\star$, et un mot $u \in \Sigma^\star$,[13] on note

$$u^{-1}L = \{w \in \Sigma^\star \mid wu \in L\}.$$

Q3. Démontrer que $(uv)^{-1}L = v^{-1}(u^{-1}L)$, quels que soient $u, v \in \Sigma^\star$ et $L \subseteq \Sigma^\star$.

Pour simplifier les notations pour les deux questions suivantes, on supposera que l'on nous donne un automate $\mathscr{A}$ reconnaissant un langage L, et dont l'automate miroir $\tilde{\mathscr{A}}$ est déterministe et émondé. On notera cette hypothèse $(\star)$. On notera $\mathscr{B} = (Q', \Sigma, \{I\}, F, \gamma)$ l'émondé déterministe de $\mathscr{A}$. On notera également $\mathscr{A} = (Q, \Sigma, I, \{f\}, \delta)$.[14]

Q4. Soit $q \in Q$ un état, et soit $u \in \Sigma^\star$ un mot. Montrer que si $q \in \gamma^\star(\{I\}, u)$, alors il existe un mot $w \in \Sigma^\star$ tel que $uw \in L$.

Q5. Montrer la propriété suivante, que l'on notera $(\♪)$:

$$\text{si } u, v \in \Sigma^\star \text{ tels que } u^{-1}L = v^{-1}L, \text{ alors } \gamma^\star(\{I\}, u) = \gamma^\star(\{I\}, v).$$

Q6. En déduire que si $\mathscr{A}$ est un automate quelconque (sans l'hypothèse $(\star)$) reconnaissant L, alors en posant $\mathscr{R}$ comme dans l'algorithme de la Fig. 2, alors $\mathscr{B}$ (défini comme dans l'algorithme) vérifie $(\♪)$ et reconnaît L.

On démontrera plus tard que $\mathscr{B}$ est minimal. On commence par implémenter cet algorithme.

Indication OCaml. Dans le code mis à disposition, vous trouverez les fonctions :
- `det : automaton -> automaton` correspondant à la fonction det ;
- `reach : automaton -> automaton` correspondant à la fonction émonde.

La fonction `transpose : automaton -> automaton` a été implémentée en Q4 de l'exercice 5.

Q7. À l'aide des fonctions implémentées précédemment et des fonctions mises à disposition, écrire une fonction `minimize_brzozowsky : automaton -> automaton` qui implémente l'algorithme en Fig. 2.

[13] Attention ! On ne choisit pas forcément $w \in L$.
[14] L'hypothèse qu'il existe un unique état final vient du déterminisme de l'automate miroir $\tilde{\mathscr{A}}$.

VI.2. Interlude : congruences sur un automate

Entre ce deux algorithmes, on introduit la notion de congruence. Elle est au cœur de l'algorithme de Moore, et permet de démontrer la correction de l'algorithme de Brzozowsky.

Dans cette section, on considère un automate $\mathscr{A} = (Q, \Sigma, \{i\}, F, \delta)$ déterministe et complet.

Définition. Une *congruence* est une relation d'équivalence $\sim$ sur Q qui vérifie, pour tout $(p, p') \in Q^2$ et tout $a \in \Sigma$:

$$(\♪) : \quad p \sim p' \iff (p \in F \iff p' \in F)$$
$$(\♫) : \quad p \sim p' \iff \delta^\star(p, a) \sim \delta^\star(p', a).$$

Définition. Étant donnés un automate $\mathscr{A}$ et une congruence $\sim$, on définit l'*automate quotient*, qui sera noté $\mathscr{A}/_\sim = (Q/_\sim, \Sigma, \mathscr{C}\ell(i), F/_\sim, \delta_\sim)$ où l'on note

$$X/_\sim = \{\mathscr{C}\ell(x) \mid x \in X\}^{[15]}$$

et l'on pose $\delta_\sim(\mathscr{C}\ell(q), a) = \mathscr{C}\ell(\delta(q, a))$,[16] sous la forme d'une fonction de transition.

Q8. Démontrer que, pour tout automate déterministe et complet $\mathscr{A}$, et toute congruence $\sim$ sur $\mathscr{A}$, on ait $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}/_\sim)$. On pourra s'aider de l'indication suivante.

Indication. On rappelle que l'on peut définir le *langage* d'un automate $\mathscr{A}$ déterministe et complet en utilisant la fonction de transition étendue, comme ci-dessous :

$$\mathscr{L}(\mathscr{A}) = \{w \in \Sigma^\star \mid \delta^\star(q_0, w) \in F_{\mathscr{A}}\},$$

où q_0 est l'*unique* état initial de $\mathscr{A}$.

Définition. On définit la *congruence de Nérode*, notée $\approx$, par :

$$\forall (p, q) \in Q_{\mathscr{A}}^2, \quad p \approx q \overset{\text{déf.}}{\iff} L_p = L_q$$

où, pour $p \in Q_{\mathscr{A}}$, $L_p = \{w \in \Sigma^\star \mid \delta^\star(p, w) \in F_{\mathscr{A}}\}$.

Q9. Démontrer que $\approx$ est bien une congruence.

D'après Q8, on sait que $\mathscr{A}/_\approx$ est équivalent à $\mathscr{A}$. Nous allons, à présent, montrer qu'il est minimal.

[15] On rappelle que $\mathscr{C}\ell$ représente la classe d'équivalence pour la relation $\sim$. Ainsi, $\mathscr{C}\ell\ x = \{y \mid x \sim y\}$.
[16] Cette définition est possible grace à la propriété $(\♫)$.