

AUTOMATES FINIS

I Automates finis déterministes

1.1 Premiers exemples

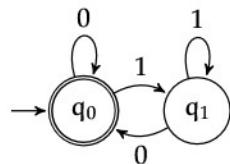


FIGURE VI.1 – Un automate qui reconnaît les représentations binaires des entiers pairs.

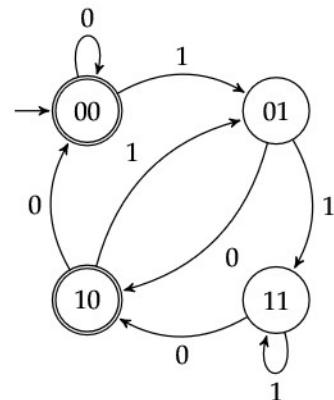


FIGURE VI.2 – Que dire de cet automate par rapport au précédent ?

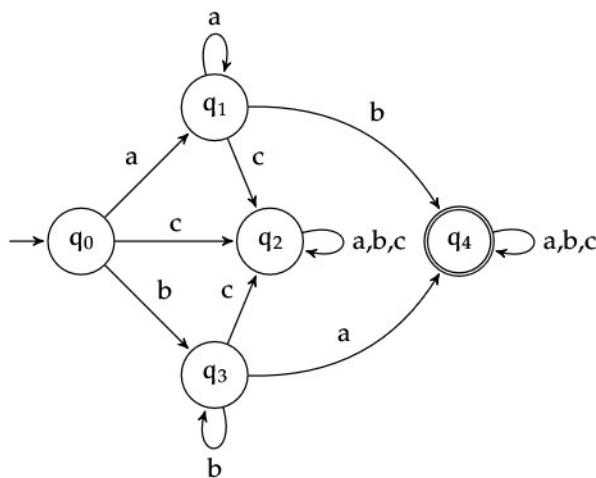


FIGURE VI.3 – Comment caractériser les mots qui nous font passer de l'état q_0 à l'état q_4 ?

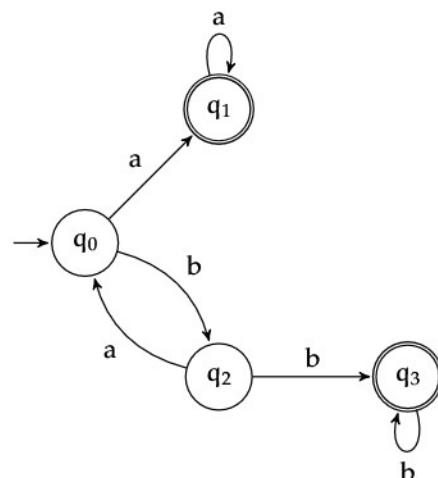


FIGURE VI.4 – Un automate non complet : si l'on essaie de suivre une flèche inexistante (en lisant b depuis l'état q_1 , par exemple), on a perdu. Quels sont les mots gagnants ?

1.2 Définitions

1.2.a Automate fini déterministe

Définition VI.1 – Automate fini déterministe

Un *automate fini déterministe* (ou AFD ou DFA) est un quintuplet $(\Sigma, Q, q_I, F, \delta)$ où :

- Σ est un alphabet;
- Q est un ensemble fini dont les éléments sont appelés *états*;
- q_I est un élément de Q appelé *état initial*;
- F est une partie de Q dont les éléments sont appelés *états finaux* (ou *états acceptants*);
- δ est une application partielle de $Q \times \Sigma$ dans Q , appelée *fonction de transition*.

Si δ est totale (*i.e.* si $\delta(q, s)$ est bien défini pour tout $(q, s) \in Q \times \Sigma$), l'automate est dit *complet*.

On peut voir (et représenter graphiquement) un automate fini comme un multi-graphe orienté :

- les sommets sont les états;
- si $\delta(q, s) = q'$, il y a un arc étiqueté par s de l'état q vers l'état q' ;
- s'il y a plusieurs arcs $q \xrightarrow{a_1} q', \dots, q \xrightarrow{a_n} q'$, on ne représentera graphiquement qu'une arête, que l'on étiquettera a_1, \dots, a_n ;
- on représente l'état initial avec une flèche entrante;
- on représente les éventuels états finaux en les entourant deux fois.

Exercice VI.1

p. 35

1. Dessiner l'automate suivant :

- $\Sigma = \{0, 1\}$
- $Q = \{A, B, C\}$
- $q_I = A$
- $F = \{C\}$

■ δ définie par

δ	0	1
A	A	B
B	C	
C	C	

2. Donner Σ, Q, q_I, F et δ pour les automates des figures VI.3 et VI.4.

1.2.b Langage d'un automate

Définition VI.2

Soit $A = (Q, q_I, F, \delta)$ un automate sur l'alphabet Σ .

- Si $a \in \Sigma$, on note $q \xrightarrow{a} q'$ pour $\delta(q, a) = q'$.
- On étend δ en une fonction partielle $\delta^* : Q \times \Sigma^* \rightarrow Q$ comme suit :
 - $\delta^*(q, \epsilon) = q$;
 - $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$ si cette expression est bien définie;
 - $\delta^*(q, ua)$ non définie dans les autres cas.

Autrement dit, on a $\delta^*(q, a_1 \dots a_n) = q'$ si la lecture successive des lettres a_1, \dots, a_n depuis l'état q nous fait arriver dans l'état q' .

- On dit que l'automate A reconnaît (ou *accepte*) le mot u si $\delta^*(q_I, u) \in F$.
- Le *langage reconnu* par l'automate A , noté $\mathcal{L}(A)$, est l'ensemble des mots reconnus par A .

Remarque

Une récurrence immédiate sur $|v|$ montre que $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$ (sous réserve de bonne définition).

Définition VI.3 – Notation alternative

Si q est un état d'un automate d'alphabet Σ et de fonction de transition δ , on note :

- $q.\varepsilon = q$;
- $q.a := \delta(q, a)$ si $a \in \Sigma$;
- $q.(u.a) := (q.u).a = \delta(q.u, a)$ si $u \in \Sigma^*$ et $a \in \Sigma$.

On a alors $q.u = \delta^*(q, u)$ pour $u \in \Sigma^*$.

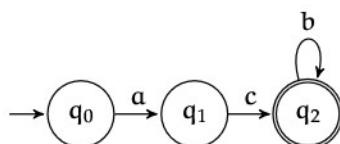
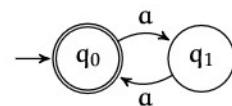
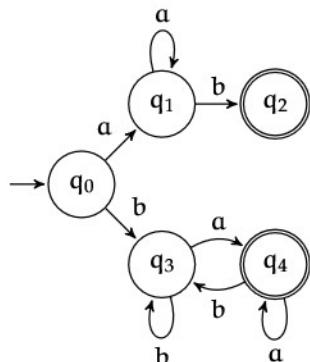
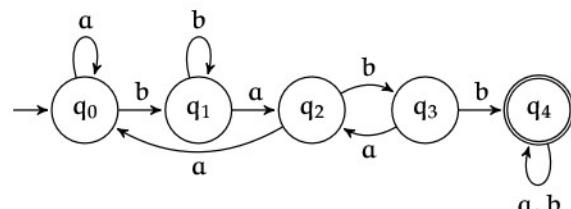
Remarques

- $q.a$ n'est défini que si $\delta(q, a)$ l'est (de même pour $q.u$).
- Cette notation est assez pratique : par exemple, en partant de q et en lisant successivement les mots u et v , on arrive en $q' = (q.u).v = q_0.(uv)\dots$

Exercice VI.2

p. 35

Pour chacun des automates suivants, donner une expression régulière décrivant le langage qu'il reconnaît.

FIGURE VI.5 – L'automate A_1 .FIGURE VI.6 – L'automate A_2 .FIGURE VI.7 – L'automate A_3 .FIGURE VI.8 – L'automate A_4 .**Exercice VI.3**

p. 36

Dessiner des automates reconnaissant les langages suivants :

1. $\Sigma = \{a, b, c\}$, L décrit par $(a|b)^*$.
2. $\Sigma = \{a, b, c\}$, $L = \{u \in \Sigma^*, |u|_a = 1 \pmod 3\}$.
3. $\Sigma = \{a, b\}$, $L = \{u \in \Sigma^*, |u|_a \pmod 2 = |u|_b \pmod 3\}$.
4. $\Sigma = \{a, b\}$, L décrit par $(a|b)^*a^2(a|b)^*$.

Définition VI.4 – Automates équivalents

Deux automates finis (sur un même alphabet) sont dits équivalents s'ils reconnaissent le même langage.

Remarque

Le rôle d'une expression est avant tout de spécifier un langage : un utilisant le vocabulaire de l'informatique théorique, on s'intéresse avant tout à leur *sémantique dénotationnelle* (i.e. à l'objet mathématique qui leur correspond). C'est pourquoi deux expressions équivalentes (comme $a(b|c)$ et $ab|ac$) peuvent presque être considérées comme égales. En revanche, pour les automates finis, on s'intéresse à la *sémantique opérationnelle*, c'est-à-dire à l'évolution de l'état de la machine au cours du calcul, et aussi au coût de ce calcul (le nombre d'états de l'automate, en particulier). Le lien entre deux automates équivalents est similaire au lien entre deux implémentations d'une même fonction mathématique dans un langage de programmation : elles n'utilisent pas forcément le même algorithme, elles n'ont pas forcément la même complexité... .

1.2.c Complétion d'un automate

Un état dont on ne peut sortir (un état q tel que $q \xrightarrow{a} q$ pour tout $a \in \Sigma$) est appelé « état puits ». À partir d'un automate quelconque, on peut obtenir un automate complet équivalent en rajoutant un nouvel état puits non terminal et en faisant pointer toutes les transitions manquantes vers cet état puits :

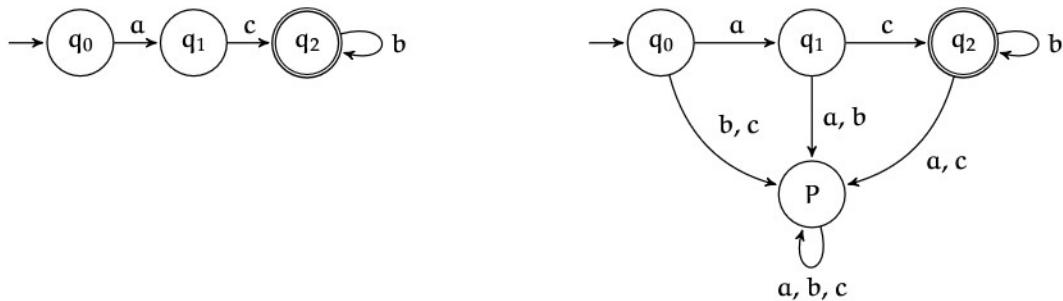


FIGURE VI.9 – Exemple de complétion d'un automate.

On en déduit la proposition suivante :

Propriété VI.5 – Complétion d'un automate

Pour tout automate fini à n états, il existe un automate complet ayant au plus $n + 1$ états qui lui soit équivalent.

Remarque

Pour alléger le dessin, on se contentera souvent d'indiquer la présence d'un état puits sans le représenter.

1.3 Langages reconnaissables**Définition VI.6 – Langage reconnaissable**

Un langage L sur un alphabet Σ est dit *reconnaissable* s'il existe un automate fini A d'alphabet Σ tel que $\mathcal{L}(A) = L$.

Remarques

- Tout langage de cardinal 0 ou 1 est clairement reconnaissable. Le prouver.
- Pour montrer qu'un langage est reconnaissable, on pourra :
 - soit exhiber directement un automate le reconnaissant;
 - soit utiliser des propriétés de clôture (comme par exemple la proposition VI.7).
- Pour montrer qu'un langage n'est *pas* reconnaissable, une technique usuelle est de montrer que sa reconnaissance nécessiterait un nombre infini d'états : l'exercice VI.4 fournit un exemple canonique de ce style de démonstration. Une autre possibilité est d'utiliser le lemme de l'étoile (VI.41).

- Essentiellement, un langage est reconnaissable si et seulement si on peut décider l'appartenance avec une complexité spatiale en $\mathcal{O}(1)$.

► Exercice VI.4

p. 36

Montrer que le langage $\{a^n b^n, n \geq 0\}$ n'est pas reconnaissable.

Propriété VI.7

L'ensemble des langages reconnaissables est clos par passage au complémentaire :

si $L \subset \Sigma^*$ est reconnaissable, alors $L^c = \{u \in \Sigma^*, u \notin L\}$ est reconnaissable.

► Exercice VI.5

p. 36

Démontrer la propriété VI.7.

Définition VI.8 – Automate produit

Soient $A = (Q, q_I, F, \delta)$ et $A' = (Q', q'_I, F', \delta')$ deux automates finis déterministes complets sur un même alphabet. Un *automate produit* B est obtenu en prenant :

- pour ensemble d'états le produit cartésien $R = Q \times Q'$;
- pour état initial l'état $r_I = (q_I, q'_I)$;
- pour fonction de transition η définie par $\eta((q, q'), a) = (\delta(q, a), \delta'(q', a))$;
- un certain ensemble $F \subset R$ d'états finaux.

Remarques

- L'idée fondamentale de l'automate produit est de simuler en parallèle l'exécution des deux automates A et A' .
- L'hypothèse de complétude sur A et A' n'est pas fondamentale, elle ne fait que simplifier un peu la définition, et assurer que l'automate produit est également complet.

Exemple VI.6

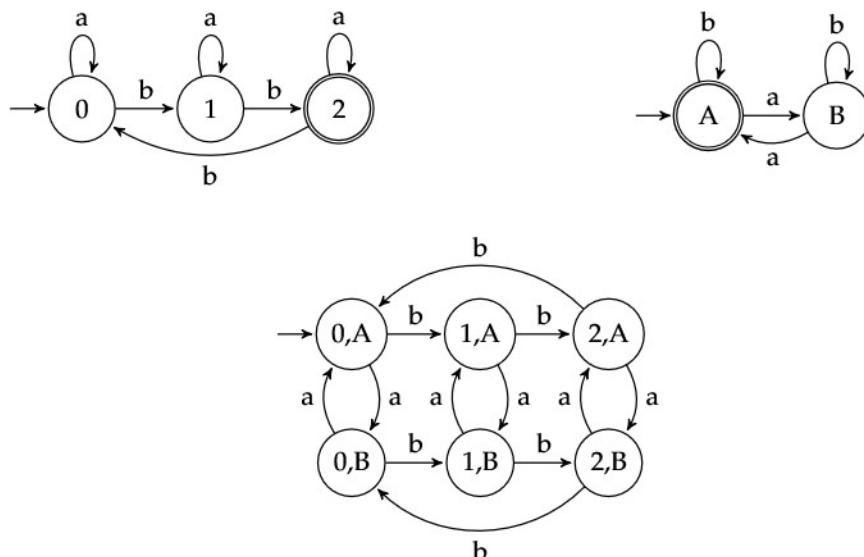


FIGURE VI.10 – Deux automates et leur produit (on n'a pas spécifié quels états étaient acceptants).

Propriété VI.9

En reprenant les notations de la définition, on a $\eta^*((q, q'), u) = (\delta^*(q, u), \delta'^*(q', u))$ pour tout mot $u \in \Sigma$ et tout couple $(q, q') \in Q \times Q'$.

Démonstration

Pour alléger, on sous-entend la fonction de transition : $q.u$ signifie $\delta^*(q, u)$, $(q, q').u$ signifie $\eta^*((q, q'), u)$...
On procède par récurrence sur la longueur de u :

- si $u = \epsilon$, on a bien $(q, q').\epsilon = (q, q') = (q.\epsilon, q'.\epsilon)$;
- sinon, $u = av$ avec $a \in \Sigma$. On a alors $(q, q').u = (q, q').(av) = (q.a, q'.a).v$ par définition.
Or $|v| = |u| - 1$, donc par hypothèse de récurrence $(q.a, q'.a).v = ((q.a).v, (q'.a).v) = (q.av, q'.av)$.
Finalement, on a bien $(q, q').u = (q.u, q'.u)$.

■

Propriété VI.10

L'ensemble des langages reconnaissables est clos par union et par intersection.

► **Exercice VI.7**

p. 36

Démontrer la propriété VI.10.

On peut légitimement se demander si les langages reconnaissables sont clos par concaténation et par étoile de Kleene, ce qui assurerait qu'ils contiennent les langages rationnels. La réponse est oui, et on a même un résultat plus fort :

Théorème VI.11 – Kleene

Les langages reconnaissables sont exactement les langages rationnels.

Autrement dit, pour tout alphabet Σ et tout langage L sur Σ , il existe une expression régulière e telle que $L = \mathcal{L}(e)$ si et seulement si il existe un automate fini déterministe A tel que $L = \mathcal{L}(A)$.

Remarques

- C'est le théorème majeur sur les automates finis et expressions régulières. Sa preuve sera l'un des principaux objectifs du reste du chapitre.
- Certaines propriétés sont très faciles à prouver pour les langages reconnaissables, d'autres pour les langages rationnels. Il sera donc assez courant de faire des aller-retours.

Exercice VI.8

p. 37

1. En admettant temporairement le théorème de Kleene, montrer que l'ensemble des mots sur $\{a, b\}$ n'admettant pas $abab$ comme facteur est rationnel.
2. Si vous n'avez rien de mieux à faire : essayer de le prouver directement.

1.4 Accessibilité

Définition VI.12 – États accessibles, co-accessibles

Soit $A = (\Sigma, Q, q_i, F, \delta)$ un automate fini déterministe. Un état q de A est dit :

- *accessible* s'il existe $w \in \Sigma^*$ tel que $q_i.w = q$;
- *co-accessible* s'il existe $w \in \Sigma^*$ et $q' \in F$ tels que $q.w = q'$.

Remarques

- Autrement dit, un état q est accessible s'il existe un chemin de q_i vers q dans le graphe sous-jacent de l'automate (le graphe obtenu en oubliant les étiquettes des transitions).
- De même, q est co-accessible s'il existe un chemin de l'un des états finaux vers q dans le graphe sous-jacent renversé (celui dans lequel on a inversé le sens de toutes les flèches).
- Il est donc relativement aisés, d'un point de vue calculatoire, de déterminer l'ensemble des états accessibles et co-accessibles (un parcours de graphe dans le premier cas, un parcours à partir de chaque état final dans l'autre).

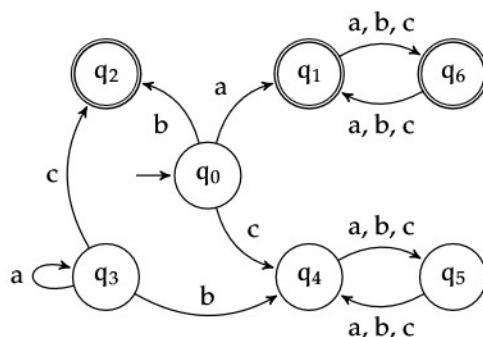


FIGURE VI.12 – L'état q_3 n'est pas accessible, les états q_4 et q_5 ne sont pas co-accessibles.

Définition VI.13 – Automate émondé

Un automate déterministe est dit *émondé* si tous ses états sont à la fois accessibles et co-accessibles.

Propriété VI.14 – Émondage

Soit A un AFD. L'automate obtenu en ne gardant que les états simultanément accessibles et co-accessibles de A , et en restreignant δ en conséquence, est un automate émondé équivalent à A .

Remarques

- Essentiellement, émonder un automate, c'est supprimer les états qui ne servent visiblement à rien.
- En général, un automate émondé n'est pas complet, et on ne peut pas le compléter en gardant le caractère émondé. Si l'on veut (presque) avoir les deux à la fois, il faut s'autoriser *un* état non co-accessible (l'état puits).
- Ce n'est pas parce qu'un automate est émondé qu'il est *minimal* : il peut tout-à-fait y avoir un autre automate, de forme *a priori* différente, qui reconnaît le même langage en utilisant moins d'états. Nous reviendrons plus tard sur le problème de la minimisation d'un automate déterministe.

Exercice VI.9

p. 37

Émonder l'automate de la figure VI.12 et montrer que l'automate obtenu n'est pas minimal.

2 Automates finis non déterministes

2.1 Premiers exemples

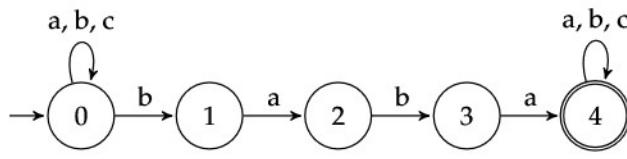
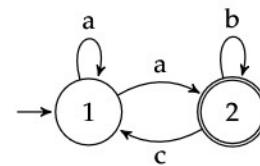
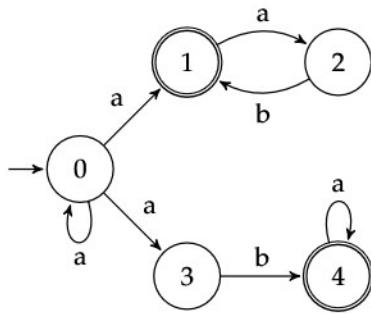
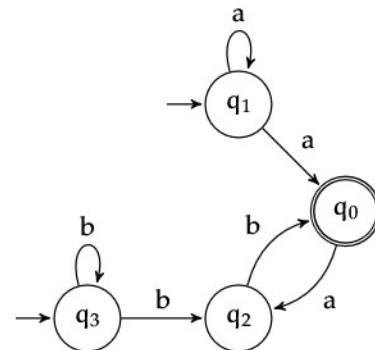


FIGURE VI.13 – L’automate baba.

FIGURE VI.14 – L’automate B_1 .FIGURE VI.15 – L’automate B_2 .FIGURE VI.16 – L’automate B_3 .

2.2 Définitions

Définition VI.15 – Automate fini non déterministe

Un *automate fini* (non déterministe) est un quintuplet $A = (\Sigma, Q, I, F, \delta)$ où :

- Σ est un alphabet (fini);
- Q est un ensemble fini d’*états*;
- $I \subset Q$, $I \neq \emptyset$ est l’ensemble des *états initiaux*;
- $F \subset Q$ est l’ensemble des *états finaux* ou *états acceptants*;
- $\delta \subset Q \times \Sigma \times Q$ est la *relation de transition*. On pourra noter $q' \in \delta(q, a)$ ou $q \xrightarrow{a} q'$ pour $(q, a, q') \in \delta$.

Remarques

- Il est clair que les automates finis déterministes peuvent être considérés comme des cas particuliers d’automates finis non déterministes.
- Les automates finis non déterministes sont donc une extension des automates finis déterministes qui autorise :
 - plusieurs états initiaux (on remplace q_1 par un ensemble $I \subset Q$);
 - plusieurs transitions partant d’un même état et étiquetées par la même lettre (δ est une relation et pas nécessairement une application partielle).
- Avec les notations de la définition, A est déterministe si :
 - $\text{Card } I = 1$;
 - la relation δ est fonctionnelle : $(q \xrightarrow{a} q' \text{ et } q \xrightarrow{a} q'' \Rightarrow q' = q'')$.
- On peut définir un automate fini non déterministe *complet* en exigeant que $\delta(q, a)$ soit systématiquement non vide, mais il ne semble pas que cela ait un quelconque intérêt.

Définition VI.I6 – Langage d'un automate fini

- On étend la fonction de transition δ aux parties de Q en posant $\delta(X, a) = \bigcup_{q \in X} \delta(q, a)$ si $X \subset Q$.
- On définit ensuite $\delta^* : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$ comme suit :
 - $\delta^*(X, \varepsilon) = X$ pour tout $X \subset Q$;
 - $\delta^*(X, ua) = \delta(\delta^*(X, u), a)$.
 Autrement dit, $\delta^*(q, w)$ est l'ensemble des états que l'on peut atteindre à partir de q en suivant des transitions étiquetées par les lettres de w (dans l'ordre).
- Un mot w est *accepté* par A s'il existe un état $q \in I$ et un état $q' \in F$ tels que $q' \in \delta^*(q, w)$, autrement dit si $\delta^*(I, w) \cap F \neq \emptyset$.
- Le *langage reconnu* par A est l'ensemble des mots reconnus par A . On le notera $\mathcal{L}(A)$.

Remarques

- Dans le cas où A se trouve être déterministe (I est un singleton, δ est une relation fonctionnelle), la fonction δ^* , la notion de mot accepté et la notion de langage reconnu coïncident bien avec celles définies pour les automates déterministes.
- À nouveau, on a facilement $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$ par récurrence sur $|v|$.

Exercice VI.I0

p. 37

Déterminer les langages reconnus par les automates de la section 2.1.

Exercice VI.II

p. 38

Construire des automates finis (non déterministes) reconnaissant les langages suivants :

1. les mots sur $\Sigma = \{a, b\}$ ayant bba comme suffixe;
2. les mots w sur $\Sigma = \{a\}$ vérifiant $|w| \equiv 1 \pmod{3}$ ou $|w| \equiv 2 \pmod{5}$;
3. les mots sur $\Sigma = \{a, b, c\}$ ayant bba ou cab comme facteur.

2.3 Déterminisation**2.3.a Automate des parties**

Pour un automate déterministe (complet, pour simplifier), on sait précisément dans quel état on se trouve après avoir lu un mot w depuis l'état initial. Pour un automate non déterministe, l'état atteint est le résultat d'une série de choix (choix de l'état initial, choix de la transition suivie à chaque étape). On ne peut donc déterminer qu'un *ensemble d'états* dans lesquels on peut se trouver après lecture de w . Cependant, cet ensemble est lui entièrement déterminé : si l'on part d'un état q et que l'on lit u , les états possibles sont précisément ceux de $\delta^*(q, u)$, (avec les notations de la définition VI.16). On peut donc transformer un automate non-déterministe ayant Q comme ensemble d'états en un automate déterministe ayant $\mathcal{P}(Q)$ comme ensemble d'états.

Définition VI.I7 – Automate des parties

Soit $A = (\Sigma, Q, I, F, \delta)$ un automate (non-déterministe). L'*automate des parties* associé à A est l'automate déterministe complet $(\Sigma, Q', q'_I, F', \eta)$ où :

- $Q' = \mathcal{P}(Q)$;
- $q'_I = I$;
- $F' = \{X \subset Q, X \cap F \neq \emptyset\}$;
- pour $X \in \mathcal{P}(Q)$ et $a \in \Sigma$, $\eta(X, a) = \delta(X, a)$.

Remarques

- Dans le dernier point, $\delta(X, a) := \bigcup_{q \in X} \delta(q, a)$ est l'extension de la fonction de transition non déterministe de A aux parties de Q . En revanche, η est une fonction de transition déterministe : à un état de A' (c'est-à-dire un ensemble d'états de A) et une lettre, elle associe un unique état de A' (c'est-à-dire à nouveau un unique ensemble d'états de A).
- Tel que présenté ici, l'automate des parties a un état pour chaque partie de Q , donc $2^{|Q|}$ états au total. En pratique, on ne construira que les états *accessibles*, et le nombre d'états sera le plus souvent nettement inférieur à cette borne. Se référer à l'exercice VI.40 pour un exemple où la borne est presque atteinte.
- L'automate des parties est complet par construction : l'état \emptyset joue le rôle d'état puits. Quand on représentera l'automate, on l'ométra presque systématiquement.

Propriété VI.18

Soit A un automate fini et A' l'automate des parties associé. On a $\mathcal{L}(A) = \mathcal{L}(A')$.

Démonstration

Une récurrence sans difficulté sur la longueur de $u \in \Sigma^*$ montre que $\eta^*(X, u) = \delta^*(X, u)$. On a ensuite :

$$\begin{aligned}
 u \in \mathcal{L}(A') &\Leftrightarrow \eta^*(q'_I, u) \in F' && \text{par définition de } \mathcal{L}(A') \\
 &\Leftrightarrow \eta^*(q'_I, u) \cap F \neq \emptyset && \text{par définition de } F' \\
 &\Leftrightarrow \eta^*(I, u) \cap F \neq \emptyset && \text{par définition de } q'_I \\
 &\Leftrightarrow \delta^*(I, u) \cap F \neq \emptyset && \\
 &\Leftrightarrow u \in \mathcal{L}(A) && \text{par définition de } \mathcal{L}(A)
 \end{aligned}$$

Théorème VI.19

Un langage est reconnaissable par un automate fini déterministe si et seulement si il est reconnaissable par un automate fini non déterministe.

Remarques

- On pourra donc simplement parler d'un « langage reconnaissable ».
- Le non déterminisme n'apporte donc pas de « puissance » supplémentaire. Il existe cependant des langages pour laquelle la reconnaissance par un automate déterministe est *coûteuse* : ils sont reconnaissables par un automate non-déterministe à n états, mais le nombre minimal d'états d'un automate déterministe les reconnaissant est de l'ordre de 2^n .

Exemple VI.12

En reprenant l'automate B_1 vu plus haut :

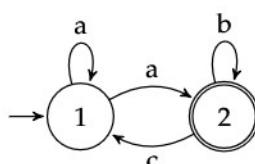


FIGURE VI.17 – L'automate B_1 .

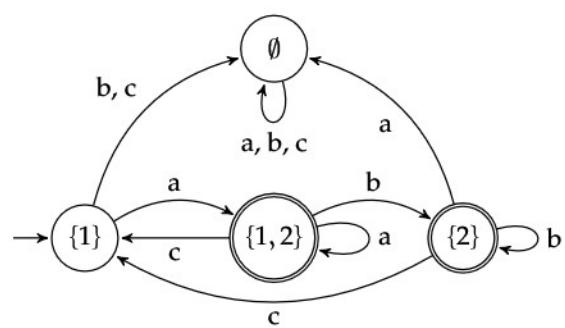


FIGURE VI.18 – L'automate des parties associé.

Exercice VI.13

p. 38

On définit :

- le *miroir* d'un mot w , noté \bar{w} , par $\begin{cases} \bar{\varepsilon} = \varepsilon \\ \overline{a_1 \dots a_n} = a_n \dots a_1 \end{cases}$

- le miroir d'un langage \mathcal{L} par $\bar{\mathcal{L}} = \{\bar{w} \mid w \in \mathcal{L}\}$

Montrer que si \mathcal{L} est reconnaissable, alors $\bar{\mathcal{L}}$ l'est également.**2.3.b Procédure pratique de déterminisation**

En pratique, on ne va construire que les états accessibles de l'automate des parties, en effectuant un parcours (en largeur dans le pseudo-code qui suit, mais cela n'a pas vraiment d'importance) de cet automate (le parcours et la construction se font simultanément). Dans l'algorithme suivant, on note $X + x$ pour $X \cup \{x\}$.

Algorithme 1 Construction de l'automate des parties

Entrée : un automate $A = (\Sigma, Q, I, F, \delta)$.

Sortie : un automate déterministe $A' = (\Sigma, Q', q'_I, F', \delta')$ tel que $L(A) = L(A')$.

Initialisation :

- $Q' \leftarrow \{I\}$

- $q'_I \leftarrow I$

- $\delta' \leftarrow \emptyset$.

- $ouverts \leftarrow \{I\}$

tant que $ouverts \neq \emptyset$ faire

 nouveaux $\leftarrow \emptyset$

pour $X \in ouverts$ faire

pour $a \in \Sigma$ faire

$Y = \bigcup_{q \in X} \delta(q, a)$

si $Y \notin Q'$ **alors**

$Q' \leftarrow Q' + Y$

 nouveaux \leftarrow nouveaux + Y

$\delta' \leftarrow \delta' + (X, a, Y)$

 ouverts \leftarrow nouveaux

$F' \leftarrow \{X \in Q', X \cap F \neq \emptyset\}$

Remarques

- Le pseudo-code est un peu vague, surtout en ce qui concerne les structures de données à utiliser.
- On ne vous demande pas de savoir programmer cet algorithme sans aide. En revanche, il faut l'avoir compris et savoir l'appliquer « à la main » sur un exemple de taille raisonnable.
- Par construction, tous les états de l'automate A' sont accessibles. Il n'y a en revanche aucune raison qu'ils soient co-accessibles.

Exercice VI.14

p. 39

Appliquer la procédure de déterminisation à l'automate de la figure VI.15.

Exercice VI.15

p. 40

Déterminiser de même l'automate de la figure VI.16.

2.4 ε -transitions

2.4.a Définitions

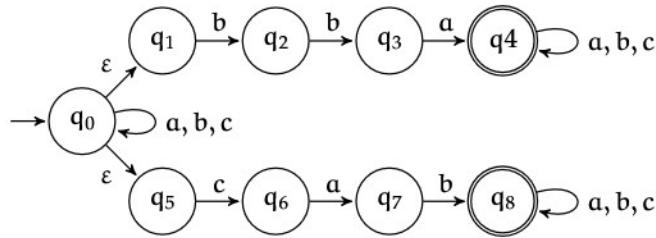


FIGURE VI.20 – Un automate sur $\Sigma = \{a, b, c\}$ reconnaissant $\Sigma^*(bba|cab)\Sigma^*$.

Définition VI.20 – Automate fini avec ε -transitions

Un *automate fini (non déterministe) avec ε -transitions* (ou *transitions spontanées*) est un quintuplet $A = (\Sigma, Q, I, F, \delta)$ où :

- Σ est un alphabet (fini) ;
- Q est un ensemble fini d'*états* ;
- $I \subset Q$, $I \neq \emptyset$ est l'ensemble des *états initiaux* ;
- $F \subset Q$ est l'ensemble des *états finaux* ou *états acceptants* ;
- $\delta \subset Q \times (\Sigma \cup \{\varepsilon\} \times Q)$ est la *relation de transition*. On pourra noter $q' \in \delta(q, a)$ ou $q \xrightarrow{a} q'$ pour $(q, a, q') \in \delta$.

Remarques

- La seule différence avec un automate fini non déterministe « basique » est que l'on peut avoir des transitions $q \xrightarrow{\varepsilon} q'$. Une telle transition, appelée ε -transition, permet de passer de l'état q à l'état q' sans consommer de lettre du mot d'entrée.
- On parlera d' ε -NFA ou ε -AFND pour un automate fini non déterministe avec ε -transitions.

2.4.b Langage d'un ε -AFND

Définition VI.21 – ε -fermeture d'un état

Soit $A = (\Sigma, Q, I, F, \delta)$ un ε -AFND. Pour un état $q \in Q$, l' ε -fermeture $E(q)$ de q est la partie de Q définie inductivement par :

Cas de base : $q \in E(q)$

Cas inductif : si $q' \in E(q)$ et $q' \xrightarrow{\varepsilon} q''$, alors $q'' \in E(q)$.

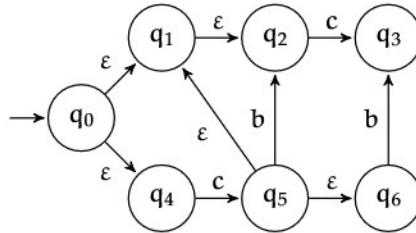
On étend cette notion aux parties de Q en posant $E(X) = \bigcup_{q \in X} E(q)$ si $X \subset Q$.

Remarques

- Comme toujours pour une définition d'ensemble inductif, $E(q)$ est le *plus petit* ensemble contenant q et clos par ε -transition.
- Autrement dit, $E(q)$ est l'ensemble des états que l'on peut atteindre à partir de q en suivant un certain nombre (éventuellement nul) d' ε -transitions.
- Une autre manière de dire la même chose est que $q' \in E(q)$ si et seulement si q' est accessible depuis q dans le graphe orienté obtenu en ne conservant que les ε -transitions de A .
- On a donc $E(E(X)) = E(X)$ pour toute partie X .
- L' ε -fermeture est un cas particulier de la notion générale de *fermeture réflexive transitive*. On pourra se référer à l'exercice VI.45.

Exercice VI.16

1. Déterminer l' ε -fermeture de chacun des états de l'automate de la figure VI.20.
2. De même pour l'automate suivant :



Pour définir le langage reconnu par un ε -AFND, il faut définir l'ensemble des états que l'on peut atteindre à partir d'un état donné en suivant des transitions dont la concaténation des étiquettes donne un certain mot. L'idée est que, lorsque l'on concatène les étiquettes des transitions, les ε -transitions sont complètement ignorées. Par exemple, un chemin $q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{a} q_3 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_5 \xrightarrow{a} q_6 \xrightarrow{\varepsilon} q_7$ est un chemin étiqueté par aba de q_0 à q_7 .

Définition VI.22 – Fonction de transition étendue

Pour un ε -AFND $A = (\Sigma, Q, I, F, \delta)$, on définit la fonction de transition étendue δ^* par :

- $\delta^*(q, \varepsilon) = E(q)$;
- pour $w \in \Sigma^*$ et $a \in \Sigma$,

$$\delta^*(q, wa) = \bigcup_{q' \in \delta^*(q, w)} E(\delta(q', a)).$$

Remarque

On peut à nouveau étendre la définition de δ et δ^* aux parties X de Q , en posant :

- $\delta(X, a) = \bigcup_{q \in X} \delta(q, a)$;
- $\delta^*(X, u) = \bigcup_{q \in X} \delta^*(q, u)$.

On a alors :

$$\delta^*(X, \varepsilon) = E(X) \text{ et } \delta^*(X, wa) = E(\delta(\delta^*(X, w), a)).$$

Exercice VI.17

Déterminer $\delta^*(q_0, \varepsilon)$, $\delta^*(q_0, b)$, $\delta^*(q_0, c)$, $\delta^*(q_0, bc)$ et $\delta^*(q_0, cb)$ pour l'automate de l'exercice VI.16.

Exercice VI.18

Les deux questions sont indépendantes.

1. A-t-on en général $\delta^*(q, aw) = E((\delta^*(\delta(q, a), w)))$?
2. Montrer que pour tous mots $u, v \in \Sigma^*$ et toute partie $X \subset Q$, on a $\delta^*(X, uv) = \delta^*(\delta^*(X, u), v)$.

Définition VI.23 – Langage d'un automate avec ε -transition

Un mot $u \in \Sigma^*$ est reconnu par un ε -AFND $A = (\Sigma, Q, I, F, \delta)$ si et seulement si $\delta^*(I, u) \cap F \neq \emptyset$. Comme pour les autres types d'automates finis, on note $\mathcal{L}(A)$ le langage reconnu par A .

Remarque

À partir d'un ε -AFND A , on peut construire un automate fini avec ε -transition équivalent A' n'ayant qu'un seul état initial. Il suffit de rajouter un nouvel état q_1 (qui sera l'unique état initial) de A' et des ε -transitions de q_1 vers chacun des états initiaux de l'automate A dont on est parti.

Exercice VI.19

Donner un automate fini avec ε -transition possédant un unique état initial et reconnaissant le langage dénoté par :

1. $a^*b^*c^*$;
2. $(a|b|c)^*(ab|aba|ca)^*$
3. $a(b|ba)^*(a|c)$

2.4.c Élimination des ε -transitions

Théorème VI.24

Tout ε -AFND est équivalent à un AFND (sans ε -transitions) possédant le même nombre d'états.

Démonstration

Soit $A = (\Sigma, Q, I, F, \delta)$ un ε -AFND. On peut alors définir $A' = (\Sigma, Q, I', F, \eta)$ comme suit :

- $I' = E(I)$;
- $\eta(q, a) = E(\delta(q, a))$.

On montre alors par récurrence sur $|u|$ que $\delta^*(I, u) = \eta^*(I', u)$.

- Si $u = \varepsilon$, alors :

$$\begin{aligned} \eta^*(I', \varepsilon) &= I' && \text{par définition de } \eta^* \\ &= E(I) && \text{par définition de } I' \\ &= \delta^*(I, \varepsilon) && \text{par définition de } \delta^* \end{aligned}$$

- Si $u = wa$ avec $a \in \Sigma$, alors :

$$\begin{aligned} \eta^*(I', wa) &= \eta(\eta^*(I', w), a) && \text{par définition de } \eta^* \\ &= \eta(\delta^*(I, w), a) && \text{par hypothèse de récurrence} \\ &= E(\delta(\delta^*(I, w), a)) && \text{par définition de } E \\ &= \delta^*(I, wa) && \text{par définition de } \delta^* \end{aligned}$$

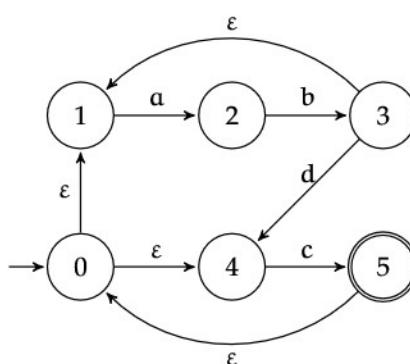
Pour tout mot u , on a donc $u \in \mathcal{L}(A) \Leftrightarrow \delta^*(I, u) \cap F \neq \emptyset \Leftrightarrow \eta^*(I', u) \cap F \neq \emptyset \Leftrightarrow u \in \mathcal{L}(A')$. ■

Remarques

- Les langages reconnaissables par un ε -AFND sont donc exactement les mêmes que ceux reconnaissables par un AFD : il suffit d'éliminer les ε -transitions puis d'utiliser la procédure de déterminisation.
- Il est tout à fait possible de réaliser ces deux étapes en même temps (passer directement d'un ε -AFND à un AFD) : écrire un tel algorithme est un bon exercice.

Exercice VI.20

Éliminer les ε -transitions de l'automate ci-dessous :



3 Automate de Thompson

Théorème VI.25 – Existence de l'automate de Thompson

Pour toute expression régulière e , il existe un ϵ -AFND $T(e)$ tel que :

- $\mathcal{L}(T(e)) = \mathcal{L}(e)$;
- $T(e)$ a exactement un état initial, et aucune transition n'arrive dans cet état initial;
- $T(e)$ a exactement un état acceptant, et aucune transition ne part de cet état acceptant.

Démonstration

Par induction structurelle sur e :



FIGURE VI.21 – $T(\emptyset)$

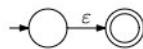


FIGURE VI.22 – $T(\epsilon)$

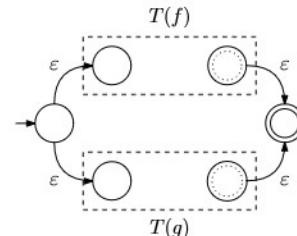


FIGURE VI.25 – $T(f|g)$

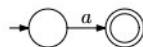


FIGURE VI.23 – $T(a)$, $a \in \Sigma$

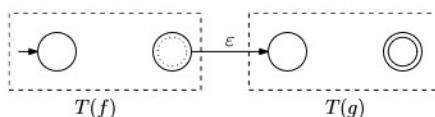


FIGURE VI.24 – $T(fg)$

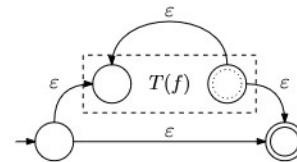


FIGURE VI.26 – $T(f^*)$

Le seul point qui n'est pas totalement évident est que l'automate $T(f^*)$ reconnaît bien $\mathcal{L}(f^*)$ (en supposant que $T(f)$ reconnaît f). On se convainc facilement que la partie supérieure reconnaît $\mathcal{L}(f)^+$, auquel on ajoute ϵ grâce à la partie inférieure de l'automate (ce n'est utile que si $\epsilon \notin \mathcal{L}(f)$). ■

Nous avons vu qu'un langage reconnaissable par un ϵ -AFND l'est également par un AFND (en éliminant les ϵ -transitions) et par un AFD (en déterminisant l'AFND obtenu à l'étape précédente). On parlera donc simplement de langage « reconnaissable », et l'existence de l'automate de Thompson permet d'obtenir le théorème suivant :

Théorème VI.26 – Kleene, sens direct

Tout langage rationnel est reconnaissable.

Exercice VI.21

p. 41

On pourrait être tenté d'utiliser une construction plus simple pour $T(f^*)$: on garde le même ensemble d'états que $T(f)$, et l'on rajoute seulement une ϵ -transition de l'état initial vers l'état final et une de l'état final vers l'état initial. La construction reste-t-elle correcte dans ce cas ?

Exercice VI.22

p. 42

1. Construire l'automate de Thompson associé à l'expression $(ab|a)^*c^*$.
2. Éliminer les ϵ -transitions de cet automate et élaguer l'automate obtenu.

4 Algorithme de Berry-Sethi

Dans cette partie, nous allons présenter une autre méthode permettant de construire un automate reconnaissant le langage associé à une expression régulière donnée. Cette méthode est moins intuitive que celle de l'automate de Thompson, mais a l'avantage de fournir directement un automate (non-déterministe) sans ϵ -transitions.

4.1 Langages locaux**4.1.a Définitions****Définition VI.27 – Ensembles P, F, S, N**

Soit L un langage sur un alphabet Σ . On définit :

- | | |
|--|--|
| ■ $P(L) = \{a \in \Sigma \mid a\Sigma^* \cap L \neq \emptyset\}$ | ensemble des premières lettres des mots de L |
| ■ $S(L) = \{a \in \Sigma \mid \Sigma^*a \cap L \neq \emptyset\}$ | ensemble des dernières lettres des mots de L |
| ■ $F(L) = \{u \in \Sigma^2 \mid \Sigma^*u\Sigma^* \cap L \neq \emptyset\}$ | ensemble des facteurs de longueur 2 des mots de L |
| ■ $N(L) = \Sigma^2 \setminus F(L)$ | ensemble des facteurs interdits de longueur 2 pour L |

Remarque

La donnée de $F(L)$ détermine entièrement $N(L)$ et inversement.

Propriété VI.28

Pour tout langage L sur un alphabet Σ , on a :

$$L \setminus \{\epsilon\} \subset (P(L)\Sigma^* \cap \Sigma^*S(L)) \setminus (\Sigma^*N(L)\Sigma^*)$$

Remarques

- L'inclusion ci-dessus signifie simplement qu'un mot non vide ne peut appartenir à L que si sa première lettre est dans $P(L)$, sa dernière lettre dans $S(L)$ et aucun de ses facteurs de longueur 2 dans $N(L)$.
- Le mot vide doit nécessairement être traité à part (il ne commence pas par une lettre de P).

En règle générale, on a seulement l'inclusion : les *langages locaux* sont ceux pour lesquels on a égalité.

Définition VI.29 – Langage local

Un langage L sur Σ est dit *local* si :

$$L \setminus \{\epsilon\} = (P(L)\Sigma^* \cap \Sigma^*S(L)) \setminus (\Sigma^*N(L)\Sigma^*)$$

Remarques

- Un langage local est essentiellement un langage pour lequel l'appartenance peut être décidée en utilisant uniquement une « fenêtre glissante » de largeur 2.
- Les langages locaux sont rationnels, mais ce n'est pas évident à partir de la définition.

Exercice VI.23

p. 43

Pour chacun des langages suivants, donner $P(L), S(L), F(L)$ et déterminer si L est local.

- | | | |
|-----------------------------|------------------------------------|----------------------------------|
| 1. $L = \mathcal{L}(abbab)$ | 3. $L = \mathcal{L}((ab)^*)$ | 5. $L = \mathcal{L}(a^* (ab)^*)$ |
| 2. $L = \mathcal{L}(a^*)$ | 4. $L = \mathcal{L}((ab)^*(cd)^*)$ | 6. $L = \mathcal{L}((ab)^*a^*)$ |

4.I.b Quelques propriétés de stabilité

Propriété VI.30

Soit L un langage local sur Σ . On a :

- L^* est local;
- $P(L^*) = P(L)$;
- $S(L^*) = S(L)$;
- $F(L^*) = F(L) \cup S(L) \cdot P(L)$.

L'exercice VI.23 montre qu'en général, ni la concaténation ni l'union de deux langages locaux n'est local. Cependant, il en va autrement si l'on considère des langages d'alphabets distincts.

Propriété VI.31

Si L et L' sont deux langages locaux sur Σ et Σ' respectivement, avec $\Sigma \cap \Sigma' = \emptyset$. On a :

- $L \cup L'$ est local;
- $P(L \cup L') = P(L) \cup P(L')$;
- $S(L \cup L') = S(L) \cup S(L')$;
- $F(L \cup L') = F(L) \cup F(L')$;
- $N(L \cup L') = N(L) \cap N(L')$.

Remarque

Les formules donnant P , S et N restent valables si les alphabets ne sont pas distincts (et même d'ailleurs si L et L' ne sont pas locaux), mais pas le caractère local de l'union (en général).

Propriété VI.32

Si L et L' sont deux langages locaux non vides sur Σ et Σ' respectivement, avec $\Sigma \cap \Sigma' = \emptyset$. On a :

- $L \cdot L'$ est local;
- $P(L \cdot L') = \begin{cases} P(L) & \text{si } \epsilon \notin L \\ P(L) \cup P(L') & \text{sinon} \end{cases}$;
- $S(L \cdot L') = \begin{cases} S(L') & \text{si } \epsilon \notin L' \\ S(L) \cup S(L') & \text{sinon} \end{cases}$;
- $F(L \cdot L') = F(L) \cup F(L') \cup S(L) \cdot P(L')$;

Remarques

- À nouveau, le fait que Σ et Σ' soient disjoints ne sert que pour obtenir le caractère local de $L \cdot L'$.
- Si L ou L' est vide, alors $L \cdot L' = \emptyset$ (qui est bien sûr local) et l'on a $P(L \cdot L') = S(L \cdot L') = F(L \cdot L') = \emptyset$.

Exercice VI.24

p. 43

Montrer que l'intersection de deux langages locaux est locale (sans condition sur les alphabets).

4.2 Expressions régulières linéaires

Comme vu plus haut, les langages rationnels ne sont pas, en général, locaux. Cependant, il en va autrement si l'on restreint toutes les opérations d'union et de concaténation à ne porter que sur

des langages d’alphabets disjoints : on introduit donc une classe restreinte d’expressions régulières formalisant cette restriction.

Définition VI.33 – Expressions régulières linéaires

Une expression régulière est dite *linéaire* si elle ne contient pas deux occurrences de la même lettre.

Remarques

- Les expressions $(ab)^*c|de$ ou $((ab)^*c)^*$ sont donc linéaires, au contraire de $(aa)^*$ et de $(ab)^*|a$.
- On peut *linéariser* une expression régulière en attribuant un numéro distinct à chacune des lettres qui y apparaissent : $(ab)^*|a$ devient $(a_1 b_2)^*|a_3$. Le langage de la nouvelle expression n’est bien sûr plus le même, mais cette opération jouera un rôle important dans la suite du chapitre.

Le théorème suivant, ainsi que sa preuve, sont importants car ils constituent une étape cruciale de la preuve du théorème de Kleene¹.

Théorème VI.34

Si e est une expression régulière linéaire sur un alphabet Σ , alors le langage de e est local.

Démonstration

Par induction structurelle sur e .

- si $e = \epsilon$, $\mathcal{L}(e) = \{\epsilon\}$ qui est local avec P, S et F vides ;
- si $e = \emptyset$, $\mathcal{L}(e) = \emptyset$ qui est local avec P, S et F vides ;
- si $e = a \in \Sigma$, alors $\mathcal{L}(e) = \{a\}$ qui est local avec $P = S = \{a\}$ et $F = \emptyset$;
- si $e = e_1^*$, alors e_1 est linéaire, donc $\mathcal{L}(e_1)$ est local par hypothèse d’induction et $\mathcal{L}(e_1)^*$ aussi d’après la proposition VI.30 ;
- si $e = e_1|e_2$, alors e_1 et e_2 sont linéaires, et les langages associés L_1 et L_2 sont donc locaux par hypothèse d’induction. De plus, les lettres qui apparaissent dans e_1 et dans e_2 sont distinctes, donc L_1 et L_2 peuvent être vus comme des langages sur des alphabets disjoints. Donc $\mathcal{L}(e) = L_1 \cup L_2$ est local d’après la proposition VI.31 ;
- le raisonnement pour $e = e_1 \cdot e_2$ est très similaire.

Remarque

La réciproque du théorème est fausse : une expression comme $a|a^*$ n’est pas linéaire mais dénote le même langage (local) que l’expression linéaire a^* . On peut même aller plus loin : le langage de l’expression aa^* est local mais n’est dénoté par aucune expression linéaire.

Il faut également noter, car cela sera crucial pour la suite, que l’on dispose d’un algorithme permettant de calculer facilement les ensembles P, S et F associés à une expression linéaire : il suffit d’appliquer récursivement les formules des propositions VI.30, VI.31 et VI.32 avec les cas de base énoncés dans la preuve ci-dessus. Cet algorithme sera implémenté au TP ??.

4.3 Automates locaux

Définition VI.35 – Automate local

Un automate fini déterministe $(\Sigma, Q, q_0, F, \delta)$ est dit *local* si, pour chaque lettre $x \in \Sigma$, toutes les transitions étiquetées par x arrivent dans un même état.

Il est de plus dit *standard* si aucune transition n’arrive à l’état initial.

Remarque

Autrement dit, un automate est local si l’état actuel ne dépend que de la dernière lettre lue.

1. Ou plutôt de la preuve *qui est au programme* : la preuve la plus classique repose sur l’automate de Thompson et ne fait pas intervenir les langages locaux.

Propriété VI.36

Tout langage local est reconnaissable par un automate local standard.

Démonstration

Soit L un langage local sur Σ . On considère $P = P(L)$, $S = S(L)$ et $F = F(L)$, et l'automate défini par :

alphabet : Σ ;

ensemble des états : $\Sigma \cup \{\varepsilon\}$, un état par lettre plus un état pour le mot vide (l'état correspond à la dernière lettre lue);

état initial : ε (au départ, on n'a lu aucune lettre);

états terminaux : S (il faut terminer par une lettre de S); pp

transitions :

- une transition $\varepsilon \xrightarrow{x} x$ pour chaque $x \in P$ (il faut obligatoirement commencer par une lettre de P);
- une transition $x \xrightarrow{y} y$ pour chaque $xy \in F$ (si l'on vient de lire x , seules les y telles que $xy \in F$ sont possibles).

Il est immédiat par construction que :

- cet automate est local et standard;
- $u = a_1 \dots a_n$ est reconnu si et seulement si $a_1 \in P$, $a_i a_{i+1} \in F$ pour tout $i \in [1 \dots n-1]$ et $a_n \in F$.

Comme L est local, l'automate construit reconnaît donc exactement $L \setminus \{\varepsilon\}$. Si $\varepsilon \notin L$, on a terminé; sinon, il suffit de rendre l'état ε final (ce qui ne modifie pas le caractère local et standard de l'automate). ■

Propriété VI.37

Tout langage associé à une expression régulière linéaire est reconnaissable par un automate local standard.

Démonstration

Découle immédiatement du théorème VI.34 et de la propriété VI.36. ■

Exercice VI.25

p. 43

Construire un automate local reconnaissant $(a|bc)^*$, et un reconnaissant $(a|b)c^*$.

4.4 Automate de Glushkov

Soit \mathcal{L} un langage rationnel et e une expression régulière le dénotant.

- On linéarise l'expression e par marquage pour obtenir une expression e' . On peut par exemple remplacer la i -ème occurrence de $x \in \Sigma$ dans e par (x, i) (que l'on notera x_i). On change donc d'alphabet, en remplaçant Σ par une partie (finie) de $\Sigma \times \mathbb{N}$.
On note φ l'application d'« oubli des marques » qui remplace chaque lettre (x, i) d'un mot sur $(\Sigma \times \mathbb{N})^*$ par la lettre $x \in \Sigma$ correspondante.
- Le langage associé à cette expression étant local, on peut obtenir un automate local standard \mathcal{A}' reconnaissant $L(e')$ (sans oublier de déterminer si $\varepsilon \in L(e')$ pour choisir si l'état initial est acceptant). Il est immédiat que $L(e) = \varphi(L(e'))$.
- On efface les marques sur les transitions de \mathcal{A}' pour obtenir un automate (*a priori* non déterministe) \mathcal{A} . Autrement dit, \mathcal{A} a les mêmes états que \mathcal{A}' (y compris l'état initial et les états acceptants), et $q \xrightarrow{i} q'$ dans \mathcal{A} si et seulement si $q \xrightarrow{(x,i)} q'$ (pour un i quelconque) dans \mathcal{A}' .
- On a clairement $L(\mathcal{A}) = \varphi(L(\mathcal{A}'))$, autrement dit $L(\mathcal{A}) = L(e) = \mathcal{L}$.

Remarques

- L'automate (non-déterministe) obtenu est appelé *automate de Glushkov*.
- Le processus complet permettant de passer d'une expression régulière à l'automate de Glushkov correspondant est appelé *algorithme de Berry-Sethi*.

- L'automate de Glushkov associé à une expression e possède $|e| + 1$ états, où $|e|$ désigne le nombre de lettres (non nécessairement distinctes) de l'expression e .
- On peut bien sûr déterminiser l'automate de Glushkov si on le souhaite. On risque cependant d'obtenir un nombre d'états de l'ordre de $2^{|e|}$.

Exemple VI.26

Construisons l'automate de Glushkov de l'expression $e = (ab|a)^*c^*$.

- L'expression linéarisée est $e' = (a_1 b_2 | a_3)^* c_4^*$.
- On utilise l'algorithme vu précédemment pour calculer $P = \{a_1, a_3, c_4\}$, $S = \{b_2, a_3, c_4\}$ et $F = \{a_1 b_2, b_2 a_3, a_3 a_1, b_2 a_1, a_3 a_3, b_2 c_4, a_3 c_4\}$, et l'on en profite pour déterminer que $\epsilon \in L(e')$.
- On obtient l'automate local suivant, dans lequel on aurait pu ne pas étiqueter les transitions :

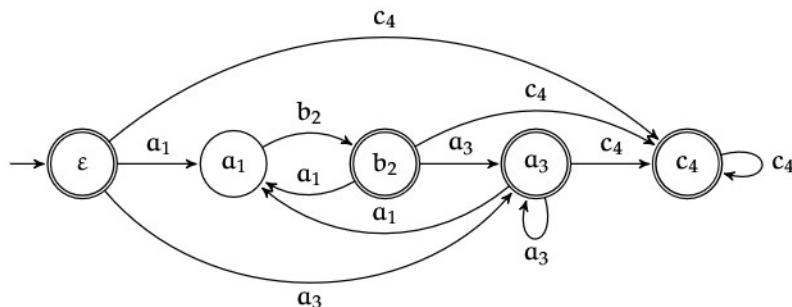


FIGURE VI.29 – L'automate local reconnaissant $L(e')$.

- On efface les marques (sur les transitions) pour obtenir l'automate de Glushkov. On peut également modifier les noms des états, mais cela n'a aucune importance.

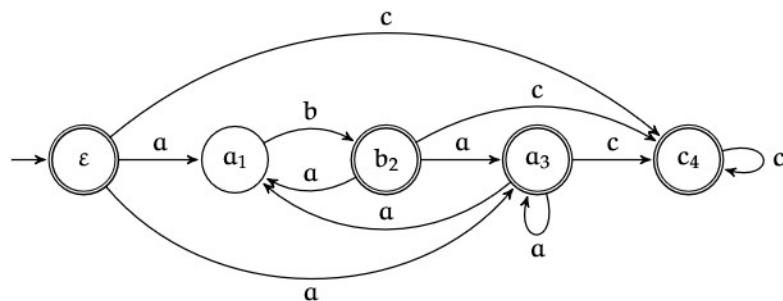


FIGURE VI.30 – L'automate de Glushkov reconnaissant $L(e)$.

- Si l'on veut un automate déterministe, on construit l'automate des parties associé.

Remarques

- L'automate de Glushkov est raisonnablement simple (son nombre d'états est $|e| + 1$), mais il n'est pas en général minimal. Ici, on peut facilement trouver un automate (et même un automate déterministe) à trois états reconnaissant $L(e)$.
- On pourra comparer à l'exercice VI.22 et constater que l'automate de Glushkov possède le même nombre d'états que celui obtenu en éliminant les ϵ -transitions puis émondant l'automate de Thompson. Voir à ce sujet l'exercice VI.41.

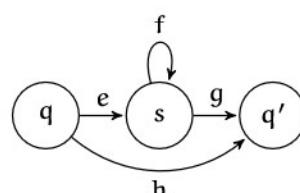
5 Sens indirect du théorème de Kleene

Il y a plusieurs façons classiques de prouver que les langages reconnaissables sont rationnels, et ces méthodes fournissent des algorithmes pour obtenir une expression régulière décrivant le langage d'un automate A donné. On notera toutefois que :

- il est beaucoup plus rarement utile de passer d'un automate à une expression que de faire l'inverse ;
- les expressions obtenues sont en général extrêmement lourdes, même quand le langage peut être décrit par une expression simple.

Une méthode (pas trop) désagréable à mettre en œuvre à la main sur de petits exemples est celle de McCluskey et Brzozowski, dite d'*élimination des états*. L'idée est de considérer des automates généralisés, dont les transitions sont étiquetées par des expressions régulières (et non seulement des lettres), et d'éliminer un par un tous les états de l'automate jusqu'à se ramener à un automate ne possédant qu'un état initial, un état final et une transition de l'un vers l'autre.

- Pour commencer, on se ramène au cas d'un automate ne possédant qu'un seul état initial, sans transition entrante. Si ce n'est pas le cas, il suffit de rajouter un nouvel état, qui sera le seul état initial, et une ϵ -transition de cet état vers chaque état initial de l'automate initial. Notons qu'ici les ϵ -transitions sont des transitions comme les autres, puisque ϵ est une expression régulière comme une autre.
- De même, on se ramène au cas où il n'y a qu'un seul état final, sans transition sortante, quitte à rajouter un nouvel état et des ϵ -transitions.
- On peut toujours considérer qu'il y a au plus une transition de chaque état vers chaque autre état : initialement, on considérera deux transitions (q, a, q') et (q, b, q') , par exemple, comme une seule transition $(q, a|b, q')$, et l'on maintiendra cette propriété tout au long de l'algorithme.
- On peut en fait considérer qu'il y a *exactement* une transition entre deux états donnés quitte à accepter des transitions étiquetées par \emptyset . En pratique on ne le fera pas, mais cela simplifie la présentation de la méthode.
- On choisit un état s différent de l'état initial et de l'état final. Pour chaque couple (q, q') où q est un prédecesseur de s et q' un successeur, on a localement la situation suivante :



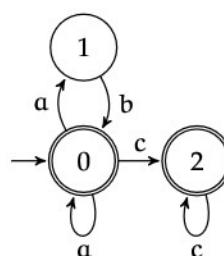
Remarque

q et q' peuvent être le même état, auquel cas la transition h est une boucle. En remplaçant la transition h par $h|ef^*g$, les transitions e et g deviennent inutiles pour passer de q à q' . Une fois que l'on a fait cette modification pour tous les couples (q, q') , on peut donc éliminer les transitions depuis et vers s , et donc l'état s lui-même.

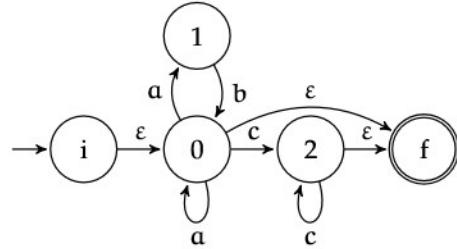
- On réitère cette procédure jusqu'à ne garder que l'état initial q_I et l'état final q_F , avec une unique transition $q_I \xrightarrow{e} q_F$. Le langage de l'automate est alors décrit par e .

Exemple VI.27

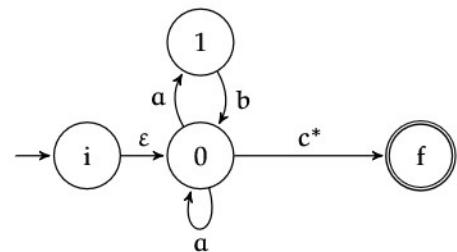
Considérons l'automate suivant :



Il y a deux états acceptants et un seul état initial, mais cet état initial possède une transition entrante : on crée donc deux nouveaux états :



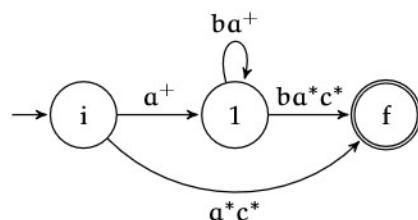
On choisit un état à éliminer, disons le numéro 2. La seule transition à modifier est celle de 0 à f, qui devient $\epsilon|cc^*\epsilon$, ce qui équivaut à c^* . On obtient donc :



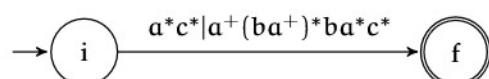
Le plus simple serait à présent d'éliminer 1, mais nous allons en fait traiter 0 en premier pour nous confronter aux difficultés. Il y a cette fois 4 couples (q, q') (avec q préédécesseur et q successeur de 0) à considérer :

- (i, f) , où la transition devient $\epsilon a^* c^* \equiv a^* c^*$;
- $(i, 1)$, où l'on obtient après simplification a^+ ;
- $(1, 1)$, de transition ba^+ ;
- $(1, f)$, de transition ba^*c^* .

On obtient :



Il ne reste plus qu'à éliminer l'état 1 et à lire l'expression sur l'automate obtenu :



6 Quelques conséquences du théorème de Kleene

6.1 Propriétés de stabilité

Propriété VI.38 – Passage au complémentaire

Les langages rationnels sont stables par passage au complémentaire. Autrement dit, si $L \subset \Sigma^*$ est rationnel, alors $\Sigma^* \setminus L$ l'est également.

Démonstration

Si L est rationnel, alors il est reconnaissable par un automate déterministe complet \mathcal{A} . Comme vu précédemment (proposition VI.7), on peut obtenir un automate reconnaissant $\Sigma^* \setminus L$ en inversant les états acceptants et non acceptants de \mathcal{A} . Ce langage est donc reconnaissable, et par conséquent rationnel.

■

Remarque

Attention, si \mathcal{A} n'est pas choisi déterministe (ou pas complet), la construction décrite n'est absolument pas valide. Si l'on part d'un automate non-déterministe \mathcal{A} et que l'on souhaite obtenir un automate reconnaissant le complémentaire de $\mathcal{L}(\mathcal{A})$, le plus simple est de commencer par déterminiser \mathcal{A} .

Propriété VI.39 – Stabilité par union des langages reconnaissables

Les langages reconnaissables sont stables par union.

Démonstration

Les langages reconnaissables sont les langages rationnels, qui sont stables par union (par construction). Circulez, il n'y a rien à voir... ■

Propriété VI.40 – Stabilité par intersection

Les langages rationnels (et reconnaissables) sont stables par intersection.

Démonstration

On note L^c pour $\Sigma^* \setminus L$. Si L_1 et L_2 sont rationnels, alors d'après les deux propriétés précédentes, il en est de même pour $(L_1^c \cup L_2^c)^c = L_1 \cap L_2$. ■

Remarques

- Il faut aussi savoir démontrer ces deux propriétés directement en utilisant un automate produit.
- D'autres propriétés de stabilité des langages rationnels sont prouvées dans les exercices de fin de chapitre, en particulier les exercices VI.46 à VI.48 qui nécessitent l'utilisation d'un automate produit. Il est conseillé de traiter au moins l'un de ces exercices.

6.2 Expressions régulières étendues

Comme on vient de prouver que les langages rationnels sont clos par intersection et passage au complémentaire, on sait qu'étant données deux expressions régulières e_1 et e_2 , il existe une expression régulière e' telle que $L(e') = L(e_1) \cap L(e_2)$ (et une expression pour $\Sigma^* \setminus L(e_1)$). On est même capable de trouver effectivement une telle expression : il « suffit » de construire un automate pour chaque expression, puis un automate pour l'intersection pour enfin en extraire une expression régulière. Cependant, ce processus est coûteux, surtout que l'expression obtenue peut être beaucoup plus complexe que celles dont on est parti. On peut donc décider d'enrichir la syntaxe des expressions régulières pour rajouter des opérateurs « \setminus » et « \cap » : on parlera d'expressions régulières étendues.

Remarque

Quand on parle en informatique « pratique » d'expressions régulières étendues, on fait référence à des extensions usuelles qui augmentent la puissance du modèle (*i.e.* qui permettent de reconnaître des langages non rationnels, comme par exemple $(\cdot +) \setminus 1$ qui reconnaît $\{u^2 \mid u \in \Sigma^+\}$). Ce n'est pas le cas ici (et, inversement, aucune syntaxe pratique pour les *regex* ne contient d'opérateurs généraux pour la différence et l'intersection).

6.3 Prouver qu'un langage n'est pas rationnel

6.3.a Lemme de l'étoile

Théorème VI.41 – Lemme de l'étoile

Soit L un langage rationnel. Il existe un entier n tel que, pour tout $u \in L$ tel que $|u| \geq n$, il existe trois mots x, y et z tels que :

- $|xy| \leq n$;
- $|y| \geq 1$;
- $\mathcal{L}(xy^*z) \subset L$.

De plus, si A est un automate reconnaissant u , alors $n = |Q|$ convient.

Remarques

- En anglais, ce théorème est connu sous le nom de *pumping lemma*; on trouve parfois l'appellation *lemme de pompage* en français.
- n est appelée *longueur de pompage* pour L (cette longueur n'est pas unique, on peut toujours la remplacer par une autre longueur plus grande).
- Dans l'énoncé, x, y et z désignent des mots, pas des lettres.
- On peut trouver un x, y et z pour chaque u assez grand : il ne s'agira pas des mêmes x, y et z pour tous les u .
- La réciproque est fausse, comme le montre l'exercice VI.51.

Démonstration

Soit A un automate (non nécessairement déterministe, mais sans ϵ -transitions) reconnaissant L , d'ensemble d'états Q avec $|Q| = n$. Soit également $u \in L$, et supposons $|u| = k \geq n$. Il y a au moins un chemin d'un état initial à un état acceptant étiqueté par u (exactement un si l'automate est déterministe) : $q_0 \xrightarrow{u_1} q_1 \dots \xrightarrow{u_k} q_n$ avec $q_0 \in I$ et $q_n \in F$. Comme $k \geq n$, on peut considérer la partie du chemin de q_0 à q_n : elle passe par $n + 1$ états, il existe donc nécessairement $i, j \in [0 \dots n]$ tels que $i < j$ et $q_i = q_j$ (autrement dit, il y a un cycle dans les n premiers arcs de ce chemin). Si l'on pose alors $x = u_1 \dots u_{i-1}$ (potentiellement vide), $y = u_i \dots u_j$ (non vide) et $z = u_{j+1} \dots u_n$, on constate que $xy^*z \in L$ pour tout $p \geq 0$: on peut parcourir le cycle un nombre quelconque de fois. On a bien $|xy| = j - i \leq n$ et $|y| = j - i \geq 1$.

■

Remarques

- Le résultat reste en fait valable si l'automate contient des ϵ -transitions mais il faut adapter la preuve, et cela n'a pas vraiment d'intérêt en pratique.
- Fondamentalement, l'idée est que si un automate contient un chemin acceptant de longueur supérieure ou égale à $|Q|$, alors il contient un cycle accessible et co-accessible, qui permet de pomper.
- Les automates ne contenant *pas* de chemin acceptant de longueur supérieure ou égale à $|Q|$ sont exactement ceux qui reconnaissent un langage de cardinal fini.

Exercice VI.28

p. 44

Montrer que le langage $P = \{a^p \mid p \text{ premier}\}$ n'est pas rationnel.

Exercice VI.29

p. 44

Soit $L = \{u \in \{a, b\}^* \mid |u|_a = |u|_b\}$. En utilisant le lemme de l'étoile, montrer que L n'est pas rationnel.

6.3.b Autour de la relation de Myhill-Nerode

Le théorème de Myhill-Nerode (et la relation du même nom) sont hors-programme. Cependant, la technique associée (montrer qu'un automate reconnaissant L aurait plus de n états, et ce pour tout n) est à connaître.

Définition VI.42 – Mots inséparables

Soit L un langage sur Σ .

- Deux mots u et v sont dits *séparés* par un mot z si exactement un des deux mots uz et vz appartient à L .
- Deux mots u et v sont dits *inséparables* si aucun mot z ne les sépare.

Remarque

En prenant $\mathcal{L} = L((ab|c)(d^*|ad))$:

- a et c sont séparés par ϵ (par exemple);
- ab et c sont inséparables;
- abd et cad sont séparés par d (et par d^k pour $k \geq 1$).

Définition VI.43 – Relation de Myhill-Nerode

Soit \mathcal{L} un langage sur Σ . La relation $\sim_{\mathcal{L}}$ définie sur Σ^* par $u \sim_{\mathcal{L}} v$ si et seulement si u et v sont inséparables est une relation d'équivalence, appelée relation de Myhill-Nerode.

Définition VI.44 – Résiduel

Soient \mathcal{L} un langage sur Σ et $u \in \Sigma^*$. Le *résiduel* (ou *quotient gauche*) de \mathcal{L} par u est l'ensemble :

$$u^{-1}\mathcal{L} = \{v \in \Sigma^* \mid uv \in \mathcal{L}\}$$

Propriété VI.45

- On a $u^{-1}\mathcal{L} = v^{-1}\mathcal{L}$ si et seulement si $u \sim_{\mathcal{L}} v$.
- Les classes d'équivalence de $\sim_{\mathcal{L}}$ sont en bijection avec l'ensemble des résiduels de \mathcal{L} .

Théorème VI.46 – Myhill-Nerode

Un langage \mathcal{L} est rationnel si et seulement si la relation de Myhill-Nerode associée possède un nombre fini de classes d'équivalences, et donc si et seulement si \mathcal{L} possède un nombre fini de résiduels.

Dans ce cas, ce nombre de classes d'équivalences est le nombre minimal d'états d'un automate fini déterministe complet reconnaissant \mathcal{L} .

Démonstration

- Supposons \mathcal{L} rationnel, et soit alors \mathcal{A} un automate déterministe complet reconnaissant \mathcal{L} . Pour q un état de \mathcal{A} , notons $P_q = \{u \in \Sigma^* \mid \delta^*(q_0, u) = q\}$ l'ensemble des mots qui nous amènent en q en partant de l'état initial. Si u et v sont dans le même P_q , alors ils sont inséparables : en effet, lire uz ou vz depuis l'état initial nous amènera dans un même état $\delta^*(q, z)$, donc uz et vz sont tous les deux acceptés ou tous les deux rejetés.

Chaque P_q est donc inclus dans l'une des classes d'équivalence, et comme la réunion des P_q est égale à Σ^* (l'automate est complet), il faut au moins un état par classe d'équivalence.

- En particulier, un langage rationnel a un nombre fini de classes d'équivalences pour la relation de Myhill-Nerode (puisque'il peut être reconnu par un automate fini).
- Inversement, soit \mathcal{L} un langage ayant un nombre fini de classes d'équivalences R_1, \dots, R_n . On définit un automate déterministe complet reconnaissant \mathcal{L} de la manière suivante :

- il y a un état pour chaque classe d'équivalence (on confondra les états et les classes d'équivalence);
- l'état initial est la classe d'équivalence de ϵ ;
- un état est final si et seulement si il contient au moins un mot de L (et si et seulement si tous ses mots sont dans L , comme on s'en convaincra facilement);
- si $u \in R$ et $a \in \Sigma$, alors $\delta(R, a)$ est l'état R' qui contient ua . Il faut vérifier que cette définition ne dépend pas du choix (arbitraire) de u , ce qui est bien le cas car u et v sont inséparables alors il en va de même pour ua et va .

Cet automate reconnaît bien \mathcal{L} et possède exactement un état par classe d'équivalence, ce qui montre que le minorant obtenu plus haut est en fait un minimum. ■

Remarques

- Le théorème n'est pas à connaître, mais il faut avoir compris la première partie de la démonstration : c'est elle que l'on adapte quand l'on souhaite démontrer qu'un langage n'est pas rationnel.
- L'automate déterministe minimal dont on vient de prouver l'existence est appelé *automate des résiduels*. Il est en fait unique à un renommage des états près (il suffit de bien regarder la démonstration), et peut être calculé de manière raisonnablement efficace à partir d'un automate déterministe quelconque reconnaissant le langage. Voir à ce sujet les exercices VI.54 et ??.
- Pour déterminer si deux automates reconnaissent le même langage, le plus efficace est de les minimiser et de déterminer si les automates obtenus sont isomorphes (mais ce n'est pas la seule solution).
- L'automate minimal est complet par construction ; il possède au maximum un état non coaccessible (état puits). Si c'est le cas, cet état peut être supprimé pour obtenir un automate déterministe (non complet) possédant un état de moins que l'automate minimal et reconnaissant le même langage.

Exercice VI.30

p. 44

Soit Σ un alphabet contenant au moins deux lettres. Montrer que le langage des palindromes sur Σ (c'est-à-dire des mots égaux à leur miroir) n'est pas rationnel.

La démonstration fournit également un moyen de décider si un automate est minimal :

Propriété VI.47

Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate déterministe complet accessible reconnaissant un langage \mathcal{L} . Les propositions suivantes sont équivalentes :

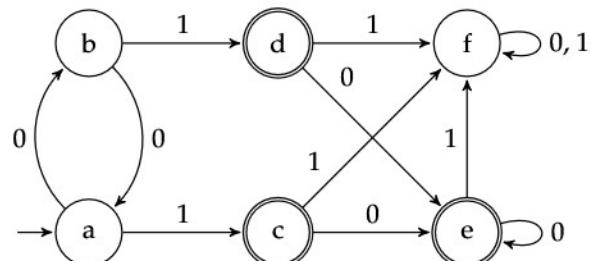
- A est minimal (parmi tous les automates déterministes complets reconnaissant \mathcal{L});
- si $q, q' \in Q$, $q \neq q'$, alors il existe $u \in \Sigma^*$ tel que $q.u \in F$ et $q'.u \notin F$ (ou inversement);
- si $q, q' \in Q$, $q \neq q'$, alors $L_q \neq L'_q$;
- si $u, v \in \Sigma^*$ sont tels que $u^{-1}\mathcal{L} = v^{-1}\mathcal{L}$, alors $q_0.u = q_0.v$.

Exercice VI.31

p. 44

1. Calculer le langage L_q associé à chacun des états q de l'automate ci-contre.

2. En déduire l'automate minimal équivalent.



Exercices

Exercices d'entraînement

Exercice VI.32

p. 45

Donner un automate pour le langage

$$\{x \in \{0, 1\}^* \mid x \text{ code en binaire un entier divisible par } 3\}$$

On commencera par une version acceptant les zéros en tête (reconnaissant donc $\epsilon, 00, 0011, 01001, 110, \dots$) puis on donnera une version n'acceptant que les représentations canoniques ($\epsilon, 11, 110, 1001, 1100, 1111, \dots$).

Exercice VI.33

p. 45

Donner une expression rationnelle correspondant au langage de l'automate suivant :

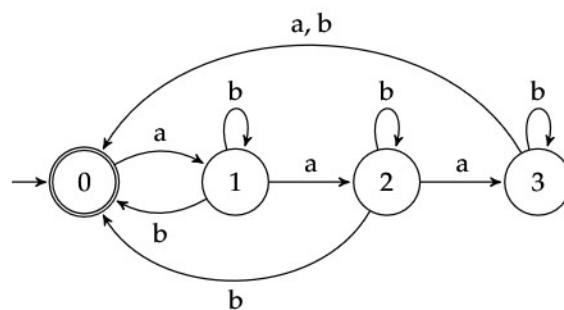


FIGURE VI.32 – L'automate A_2 .

Exercice VI.34

Pour un langage L , on note $\text{pref}(L)$ l'ensemble des préfixes des mots de L , $\text{ suff}(L)$ l'ensemble des suffixes, $\text{fact}(L)$ l'ensemble des facteurs. Montrer que si L est rationnel, ces trois langages le sont également.

Exercice VI.35

Déterminer l'automate de Thompson associé à l'expression régulière $(a|c)^*(abb|\epsilon)$ puis éliminer ses ϵ -transitions.

Exercice VI.36

Montrer que si L est local, alors $\text{fact}(L)$ (ensemble des facteurs des mots de L) l'est également.

Exercice VI.37

Déterminer l'automate de Glushkov associé à l'expression régulière $(a|c)^*(abb|\epsilon)$.

Exercice VI.38

Reprendre l'exercice VI.33 en utilisant la méthode d'élimination des états.

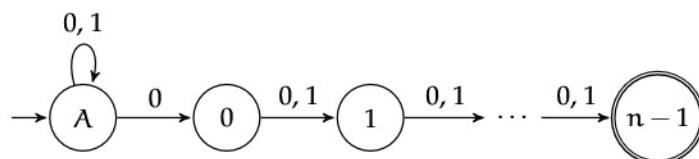
Exercice VI.39

Montrer que le langage $\mathcal{P} = \{a^p \mid p \text{ premier}\}$ n'est pas rationnel.

Autres exercices**► Exercice VI.40 – Déterminisation explosive**

p. 45

On considère l'automate \mathcal{A}_n suivant :



1. Quel est le langage reconnu par cet automate ?
2. Appliquer la procédure de déterminisation à cet automate pour $n = 2$.
3. En faire de même pour $n = 3$.
4. Combien d'états obtient-on dans le cas général quand on détermine cet automate ?
5. Montrer que l'automate déterminisé obtenu est minimal, c'est-à-dire qu'aucun automate déterministe possédant strictement moins d'état ne reconnaît le même langage.

► Exercice VI.41 – Propriétés de l'automate de Thompson

p. 46

Dans cet exercice, on ne considère que des expressions régulières ne faisant pas intervenir le symbole \emptyset .

1. Pour une expression régulière e , on note $\text{taille}(e)$ la taille de e en tant qu'arbre binaire (nombre total de noeuds, internes ou non). Majorer (de manière raisonnablement précise) le nombre d'états de l'automate de Thompson associé à e en fonction de $|e|$.
2. Les états de l'automate de Thompson sont-ils tous accessibles ? co-accessibles ?
3. On considère à présent $T(e)$ l'automate de Thompson associé à une expression e et $T'(e)$ l'automate obtenu en éliminant les ε -transitions. Quels états de $T'(e)$ sont accessibles ? co-accessibles ?
4. On note $T''(e)$ l'automate obtenu en émondant $T'(e)$. Exprimer le nombre d'états de $T''(e)$ en fonction du nombre $|e|$ de lettres (non nécessairement distinctes) qui apparaissent dans l'expression e .

► Exercice VI.42

p. 47

On considère un alphabet Σ et des langages L, L' sur Σ . Démontrer les propositions suivantes si elles sont vraies, donner un contre-exemple sinon.

1. Si L est rationnel et $L \cap L'$ ne l'est pas, alors L' n'est pas rationnel.
2. Si L est rationnel et si L' ne l'est pas, alors $L \cap L'$ n'est pas rationnel.
3. Si L est rationnel, alors L^* est rationnel.
4. Si L^* est rationnel, alors L est rationnel.
5. Si L et L' sont rationnels, alors $L \setminus L'$ l'est également.
6. Si LL' est rationnel, alors L est rationnel ou L' est rationnel.

► Exercice VI.43

Pour chacun des problèmes suivants, donner les grandes lignes d'un algorithme permettant de le résoudre. On ne rentrera pas dans les détails, mais on essaiera de donner une solution raisonnablement simple et efficace.

1. Étant donnés un mot t une expression régulière, déterminer si le mot appartient au langage dénoté par l'expression.
2. Déterminer si le langage reconnu par un automate est fini.
3. Déterminer si le langage dénoté par une expression régulière est fini.
4. Déterminer si le langage dénoté par une expression rationnelle e dénote \emptyset .
5. Déterminer si le langage reconnu par un automate est vide.
6. Déterminer si une expression régulière sur l'alphabet Σ dénote le langage Σ^* .
7. Déterminer si deux expressions régulières dénotent le même langage.

Exercice VI.44 – Fermeture arrière

La méthode d'élimination des ε -transitions donnée dans le cours est dite *par fermeture avant*. On peut aussi procéder par *fermeture arrière* en posant $\eta(q, a) = \delta(E(q), a)$. Que faut-il alors choisir comme ensemble d'états initiaux et acceptants pour obtenir un automate équivalent? On rédigera la démonstration d'équivalence.

Exercice VI.45 – Autour de la fermeture transitive

Définition VI.48 – Fermeture transitive d'une relation

Soit X un ensemble et $\mathcal{R} \subset X \times X$ une relation.

-
- On définit \mathcal{R}^n pour $n \geq 0$ par :

$$\begin{cases} x\mathcal{R}^0y \Leftrightarrow x = y \\ x\mathcal{R}^{n+1}y \Leftrightarrow \exists z \in X, x\mathcal{R}z \text{ et } z\mathcal{R}^n y \end{cases}$$

- On appelle *fermeture transitive* de \mathcal{R} la relation

$$\mathcal{R}^+ = \bigcup_{n \geq 1} \mathcal{R}^n$$

- On appelle *fermeture transitive réflexive* de \mathcal{R} la relation

$$\mathcal{R}^* = \bigcup_{n \geq 0} \mathcal{R}^n$$

1. a. On considère la relation \mathcal{R} sur \mathbb{Z} définie par $n\mathcal{R}p$ si et seulement si $p = n + 1$. Donner une caractérisation simple de \mathcal{R}^+ et \mathcal{R}^* .
- b. On considère la relation \mathcal{P} sur \mathbb{N}^* définie par :

$$n\mathcal{P}p \iff \exists k, r \in \mathbb{N}, p = 2^k n + r \text{ et } r < 2^k$$

Déterminer une relation \mathcal{R} sur \mathbb{N}^* vérifiant les deux propriétés suivantes :

- (1) : $\mathcal{P} = \mathcal{R}^*$;
 - (2) : $\forall n \in \mathbb{N}^*, \text{Card}\{p \in \mathbb{N}^* \mid n\mathcal{R}p\} = 2$.
2. Montrer que la fermeture transitive d'une relation \mathcal{R} est la plus petite relation transitive contenant \mathcal{R} .

3. On considère un graphe non orienté $G = (V, E)$ et la relation \mathcal{R} sur V correspondante (i.e $x\mathcal{R}y \iff \{x, y\} \in E$). Quelle est la structure du graphe $G' = (V, E')$ avec $\{x, y\} \in E' \iff (x \neq y \text{ et } x\mathcal{R}^+y)$?
4. On considère à présent un graphe orienté $G = (V, E)$ et sa relation d'adjacence : $x\mathcal{R}y \iff (x, y) \in E$.
 - a. Donner une condition nécessaire et suffisante simple sur G pour que \mathcal{R}^* soit une relation d'ordre.
 - b. Même question pour que \mathcal{R}^* soit une relation d'ordre *totale*.
 - c. On considère un ordre topologique \preceq sur G . Que peut-on dire de la relation \preceq par rapport à la relation \mathcal{R}^* ?

Exercice VI.46 – Racine carrée d'un langage

p. 48

Étant donné un langage L sur Σ , la *racine carrée* de L est le langage défini par

$$\sqrt{L} = \{u \in \Sigma^* \mid uu \in L\}$$

Pour un automate fini déterministe complet $A = (Q, \Sigma, q_0, F, \delta)$, on considère les automates finis A'_q (pour $q \in Q$) tels que :

- les états de A'_q sont les couples $(q_i, q_j) \in Q^2$;
 - l'état initial de A'_q est (q_0, q) ;
 - les états terminaux de A'_q sont les (q, q_f) avec $q_f \in F$;
 - tous les A'_q ont la même fonction de transition δ' définie par $\delta'((q_i, q_j), a) = (\delta(q_i, a), \delta(q_j, a))$ (pour $a \in \Sigma$);
1. Donner une caractérisation simple du langage $\mathcal{L}(A'_q)$.
 2. En déduire que la racine carrée d'un langage rationnel est rationnelle.

Exercice VI.47 – Variation

On définit $\frac{1}{2}L = \{u \in \Sigma^* \mid \exists v \in \Sigma^*, |v| = |u| \text{ et } uv \in L\}$. Montrer par une construction similaire à celle de l'exercice précédent que si L est rationnel, alors $\frac{1}{2}L$ l'est également.

Exercice VI.48 – Produit de mélange

On définit le *produit de mélange* (*shuffle product* en anglais) de deux mots u et v sur Σ par

$$u * v = \{u_1 \cdot v_1 \cdot \dots \cdot u_n \cdot v_n \mid \text{les } u_i \text{ et } v_i \text{ sont des mots tels que } u = u_1 \cdot \dots \cdot u_n \text{ et } v = v_1 \cdot \dots \cdot v_n\}$$

On remarquera bien que rien n'empêche certains des u_i et v_i d'être vides.

On étend cette définition aux langages en posant :

$$L * L' = \bigcup_{u \in L \text{ et } v \in L'} u * v$$

1. Déterminer $aa * cd$.
2. Déterminer $\text{Card}(u * v)$ en fonction de $|u|$ et $|v|$ dans le cas où les lettres de u et de v sont deux à deux distinctes.
3. Déterminer $\mathcal{L}(a^*) * \mathcal{L}(b^*)$ et $\mathcal{L}(a^*) * \mathcal{L}(bc)$.
4. Montrer que si L et L' sont rationnels, alors $L * L'$ l'est également.

Exercice VI.49 – Quotient gauche d'un langage rationnel

Pour deux langages L et K sur un même alphabet Σ , on définit :

$$K^{-1}L = \{v \in \Sigma^* \mid \exists u \in K, uv \in L\}$$

Montrer que si L est rationnel, alors $K^{-1}L$ l'est également (sans hypothèse sur K).

Exercice VI.50 – Nombres de Mersenne

Un *nombre de Mersenne* est un nombre premier de la forme $2^n - 1$. On note L le langage sur $\{0, 1\}^*$ dont les mots sont les écritures binaires des nombres premiers. En supposant qu'il existe une infinité de nombres de Mersenne^a, montrer que L n'est pas rationnel.

a. On n'en sait rien en fait, même si l'on conjecture que c'est vrai.

Exercice VI.51

Montrer que le langage L défini ci-dessous n'est pas rationnel, mais qu'il vérifie le lemme de l'étoile :

$$L = \{ab^r c^r \mid r \geq 0\} \cup \{a^r b^s c^t \mid r \neq 1, s \geq 0, t \geq 0\}.$$

Exercice VI.52 – Le barman aveugle avec des gants de boxe

On considère un barman aveugle portant des gants de boxe. Il a devant lui une table sur laquelle sont posés quatre verres (vides). Étant aveugle, il ne voit pas si les verres sont posés à l'endroit ou à l'envers; portant des gants de boxe, il n'est pas non plus capable de distinguer leur orientation au toucher. Cependant, il est suffisamment dexter pour choisir l'un des verres et le retourner. Son but est de faire en sorte que les quatre verres aient la même orientation (soit tous à l'endroit, soit tous à l'envers). Malheureusement pour lui^a, le plateau de la table est rotatif, et un « ami »^b s'amuse à le tourner après chaque tentative. Plus précisément :

- à chaque étape, le barman peut retourner autant de verres qu'il le souhaite;
- quand il a terminé, il le signale :
 - si tous les verres ont la même orientation, il est déclaré vainqueur;
 - sinon, son ami effectue une rotation du plateau (d'un nombre arbitraire de quarts de tour, y compris zéro), et on recommence.

Pouvez-vous prouver que, tout compte fait, le barman s'en sort quand même mieux que Tantale ou Sisyphe ?

Plus précisément, on demande de prouver l'existence d'une stratégie permettant au barman de gagner en au plus N coups, où N est un entier que l'on précisera.

a. On peut légitimement s'interroger sur ce qu'il a bien pu faire dans une vie antérieure pour mériter un tel supplice.

b. *With friends like these, who needs enemies?*

Problèmes

Minimisation

Exercice VI.53 – Morphisme d'automates et automate minimal

Soient $A = (\Sigma, Q_A, i_A, F_A, \delta_A)$ et $B = (\Sigma, Q_B, i_B, F_B, \delta_B)$ deux automates finis déterministes complets. Un *morphisme d'automates* de A vers B est une application $\varphi : Q_A \rightarrow Q_B$ qui vérifie les conditions suivantes :

- φ est surjective ;
- $\varphi(i_A) = i_B$;
- $\forall q \in Q_A, \varphi(q) \in F_B \Leftrightarrow q \in F_A$;
- $\forall q \in Q, \forall a \in \Sigma, \varphi(\delta_A(q, a)) = \delta_B(\varphi(q), a)$.

1. Montrer que s'il existe un morphisme de A vers B , alors A et B reconnaissent le même langage.
2. Pour $q \in Q_A$, on note $L_q = \{u \in \Sigma^* \mid \delta_A^*(q, u) \in F_A\}$. On définit alors une relation d'équivalence \sim sur Q_A par :

$$\forall q, q' \in Q_A, q \sim q' \Leftrightarrow L_q = L_{q'}.$$

On note $E = Q/\sim$ l'ensemble des classes d'équivalence de la relation \sim et $\varphi : Q_A \rightarrow E$ l'application qui à un état associe sa classe d'équivalence.

- a. Construire un automate M tel que φ soit un morphisme d'automates de A vers M .
- b. Montrer que l'automate M est minimal.

Exercice VI.54 – Algorithme de Moore

L'exercice précédent montre que le problème de la minimisation d'un automate déterministe se réduit à celui du calcul des classes d'équivalence de ses états pour la relation \sim . L'algorithme de Moore fournit une méthode efficace pour effectuer ce calcul.

On considère un automate déterministe complet $A = (\Sigma, Q_A, i_A, F_A, \delta_A)$.

1. Pour $h \geq 0$ et $q \in Q_A$, on définit le langage

$$L_q^{(h)} = \{u \in \Sigma^* \mid q.u \in F_A \text{ et } |u| \leq h\}$$

et la relation d'équivalence \sim_h sur l'ensemble des états de A par :

$$q \sim_h q' \text{ si et seulement si } L_q^{(h)} = L_{q'}^{(h)}.$$

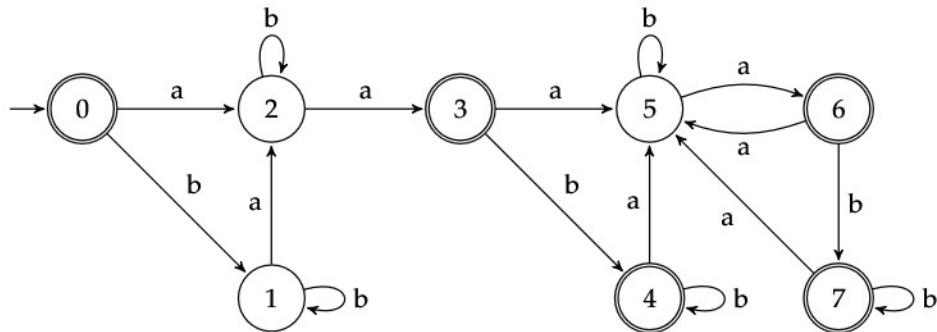
On rappelle que la relation d'équivalence \sim est définie par :

$$q \sim q' \text{ si et seulement si } L_q = L_{q'}.$$

- a. Quelles sont les classes d'équivalence de \sim_0 ?
b. Justifier que :
$$q \sim_{h+1} q' \text{ si et seulement si } q \sim_h q' \text{ et } \forall a \in \Sigma, q.a \sim_h q'.a$$

c. Montrer que si les relations \sim_h et \sim_{h+1} sont égales pour un certain h , alors les relations \sim_h et \sim sont égales.
2. On suppose à présent que les états de A sont numérotés de 0 à $n-1$. On suppose également que les classes d'équivalence de \sim_h sont numérotées de 0 à $k-1$ (pour un certain $k \leq n$) et que l'on dispose d'un tableau φ_h de longueur n tel que $\varphi_h(i)$ soit le numéro de la classe d'équivalence de l'état i pour la relation φ_h .

- a. Expliquer comment calculer le tableau φ_{h+1} en temps $O(|\Sigma| n \log n)$ à partir du tableau φ_h .
 - b. Optionnel : montrer que ce calcul peut en fait se faire en temps $O(n |\Sigma|)$.
 3. En déduire un algorithme permettant de minimiser un automate déterministe en temps $O(nls)$, où :
 - n est le nombre d'états de l'automate initial ;
 - l est le nombre d'états de l'automate minimal ;
 - s est la taille de l'alphabet.
- On pourra supposer que la fonction de transition de l'automate initial s'exécute en temps constant, et que l'on a réussi à se débarrasser du facteur logarithmique à la question précédente.
4. Appliquer cet algorithme à l'automate ci-dessous.



Exercice VI.55 – Automate minimal de Brzozowski

Pour un automate (*a priori* non déterministe) $A = (\Sigma, Q, F, I, \delta)$, on note :

- $\mathcal{T}(A) = (\Sigma, Q, F, I, \delta')$ l'automate « miroir » (ou *transposé*) de A , où $(q, a, q') \in \delta'$ si et seulement si $(q', a, q) \in \delta$ (autrement dit, $\mathcal{T}(A)$ est l'automate obtenu en échangeant le rôle des états initiaux et finals dans A et en inversant le sens de toutes les flèches) ;
 - $\mathcal{D}(A)$ l'automate des parties associé à A , dans lequel on n'a construit que les états accessibles (ce qui correspond à suivre l'algorithme de construction donné dans le cours).
1. Montrer que si A est déterministe accessible, alors $\mathcal{D}(\mathcal{T}(A))$ est un automate minimal.
 2. En déduire un algorithme prenant en entrée un automate fini (non nécessairement déterministe) A et calculant un automate déterministe complet minimal A' reconnaissant le même langage que A .

Remarque

La complexité dans le pire des cas de cet algorithme est exponentielle en le nombre d'états de l'automate initial, même lorsque celui-ci est déterministe. Cependant, les performances pratiques sont assez bonnes (les cas pathologiques sont très rares).

Exercices d'oral

Exercice VI.56 – Énumérations – Ulm 2018

On fixe un automate déterministe A sur un alphabet Σ et un entier $m \in \mathbb{N}^*$.

1. a. Proposer un algorithme qui calcule la longueur minimale des mots acceptés par A , si cette quantité est bien définie, et qui détecte si elle ne l'est pas. Quelle est sa complexité ?
- b. On suppose dans cette question que A n'a pas de cycle. Proposer un algorithme qui calcule la longueur maximale des mots acceptés par A , si cette quantité est définie, et

- qui détecte si elle ne l'est pas. Quelle est sa complexité ?
- c. Que se passe-t-il si A a un cycle ?
2. a. Proposer un algorithme permettant de calculer le nombre modulo m de mots de longueur n dans $\mathcal{L}(A)$. Quelle est sa complexité en temps et en espace ?
 - b. Le problème devient-il plus difficile si l'on demande de calculer le résultat exact (sans modulo) ?
 - c. L'hypothèse que A est déterministe est-elle importante ?
-

Exercice VI.57 – Automates et palindromes – Ulm 2018

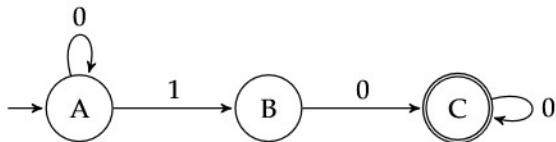
On fixe un alphabet Σ avec $|\Sigma| > 1$. Pour un mot $u = u_0 \dots u_{n-1} \in \Sigma^*$, on note $\bar{u} = u_{n-1} \dots u_0$ le miroir de u ; on dit que u est un palindrome si $u = \bar{u}$. On note Π l'ensemble des palindromes de Σ^* .

1. Soit $n \geq 0$ et $\Pi_n = \Pi \cap \Sigma^n$. Montrer que si A est un automate déterministe complet tel que $\mathcal{L}(A) \cap \Sigma^{2n} = \Pi_{2n}$, alors A possède au moins $|\Sigma|^n$ états.
 2. En déduire que Π n'est pas rationnel.
 3. Étant donné un automate fini A d'alphabet Σ , peut-on calculer un automate A_Π tel que $\mathcal{L}(A_\Pi) = \mathcal{L}(A) \cap \Pi$?
 4. Étant donné A , peut-on calculer un automate A'_Π qui reconnaisse $\{u \in \Sigma^* \mid u\bar{u} \in \mathcal{L}(A)\}$?
 5. On appelle Π_{pair} l'ensemble des palindromes de longueur paire. Proposer un algorithme qui, étant donné un automate fini A sur Σ , détermine si $\mathcal{L}(A) \cap \Pi_{\text{pair}}$ est vide, fini, ou infini. Discuter de sa complexité en temps et en espace.
 6. Modifier l'algorithme de la question précédente pour calculer la cardinalité de $\mathcal{L}(A) \cap \Pi_{\text{pair}}$ quand cet ensemble est fini, en faisant l'hypothèse que l'automate d'entrée A est déterministe. Comment la complexité est-elle affectée ?
 7. Modifier l'algorithme des deux questions précédentes pour qu'il s'applique à $\mathcal{L}(A) \cap \Pi$.
-

Solutions

Correction de l'exercice VI.1 page 2

1.



2. Pour Σ , on suppose que l'alphabet ne contient aucune lettre n'apparaissant sur aucune transition. On a alors :

Figure VI.3

- $\Sigma = \{a, b, c\}$
- $Q = \{q_0, \dots, q_4\}$
- $q_1 = q_0$
- $F = \{q_4\}$

- δ définie par :

δ	a	b	c
q_0	q_1	q_3	q_2
q_1	q_1	q_4	q_2
q_2	q_2	q_2	q_2
q_3	q_4	q_3	q_2
q_4	q_4	q_4	q_4

Figure VI.4

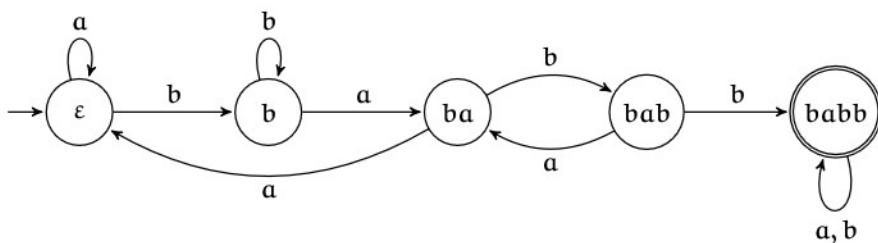
- $\Sigma = \{a, b\}$
- $Q = \{q_0, \dots, q_3\}$
- $q_1 = q_0$
- $F = \{q_1, q_3\}$

- δ définie par :

δ	a	b
q_0	q_1	q_2
q_1	q_1	
q_2	q_0	q_3
q_3		q_3

Correction de l'exercice VI.2 page 3

1. Immédiat : acb^* .
2. Immédiat : $(a^2)^*$.
3. Les mots menant à q_2 sont ceux de la forme a^+b . Pour ceux menant à q_4 , il faut passer de q_0 à q_3 (b), puis boucler un certain nombre de fois sur q_3 en passant éventuellement par q_4 (forme $(b|a^+b)^*$), puis finalement lire un a pour finir en q_4 . Finalement, on obtient $a^+b|b(b|a^+b)^*a$.
4. Pour comprendre comment fonctionne cet automate, le plus simple est de renommer les états :



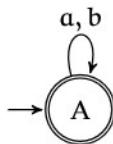
En notant $f(u)$ le plus long préfixe de $babb$ qui soit un suffixe de u , on peut alors montrer par récurrence sur $|u|$ que :

- si $babb$ est un facteur de u , alors $q_0.u = babb$;
- sinon, $q_0.u = f(u)$.

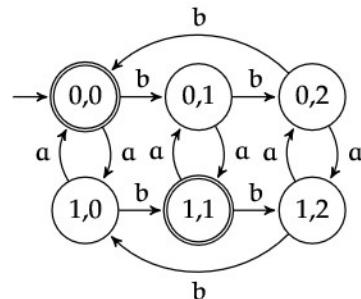
Autrement dit, on essaie de lire un facteur $babb$, et l'on boucle dans l'état $babb$ (final) dès que l'on a réussi. Ainsi, le langage reconnu est celui dénoté par $(a|b)^*babb(a|b)^*$.

Correction de l'exercice VI.3 page 3

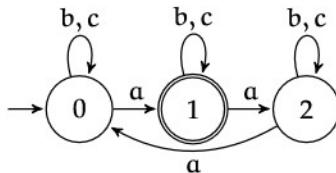
1.



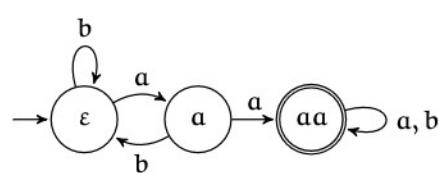
3.



2.



4.

Correction de l'exercice VI.4 page 5

Supposons $L = \{a^n b^n, n \geq 0\}$ reconnu par un automate $\mathcal{A} = (Q, q_0, F, \delta)$ (que l'on peut supposer complet). Q est fini, donc l'application $\varphi : \mathbb{N} \rightarrow Q$ définie par $\varphi(n) = q_0.a^n$ n'est pas injective. Par conséquent, il existe $i, j \in \mathbb{N}, i \neq j$ tels que $q_0.a^i = q_0.a^j$. On en déduit $q_0.a^i b^i = q_0.a^j b^i$: c'est absurde puisque $q_0.a^i b^i \in F$ et $q_0.a^j b^i \notin F$. Donc L n'est pas reconnaissable.

Correction de l'exercice VI.5 page 5

Soit L reconnaissable et $\mathcal{A} = (Q, q_I, F, \delta)$ un automate déterministe tel que $\mathcal{L}(\mathcal{A}) = L$. D'après la propriété VI.5, on peut supposer \mathcal{A} complet. Considérons alors $\mathcal{A}' = (Q, q_I, Q \setminus F, \delta)$ l'automate obtenu en échangeant les états finaux et non finaux de \mathcal{A} . Comme \mathcal{A} et \mathcal{A}' sont complets, $q_I.u$ est défini pour tout mot $u \in \Sigma^*$, et égal dans les deux automates par construction. On a :

$$u \in \mathcal{L}(\mathcal{A}) \Leftrightarrow q_I.u \in F \Leftrightarrow q_I.u \notin Q \setminus F \Leftrightarrow u \notin \mathcal{L}(\mathcal{A}')$$

Remarque

Il faut bien partir d'un automate complet pour éviter le cas où $q_I.u$ n'est pas défini.

Correction de l'exercice VI.7 page 6

Soient L, L' deux langages reconnaissables sur l'alphabet Σ et $\mathcal{A}, \mathcal{A}'$ deux automates déterministes complets les reconnaissant. On construit deux automates produit \mathcal{A}_U et \mathcal{A}_\cap en suivant la définition VI.8 et en posant $F_U = (F \times Q') \cup (Q \times F')$ et $F_\cap = F \times F'$.

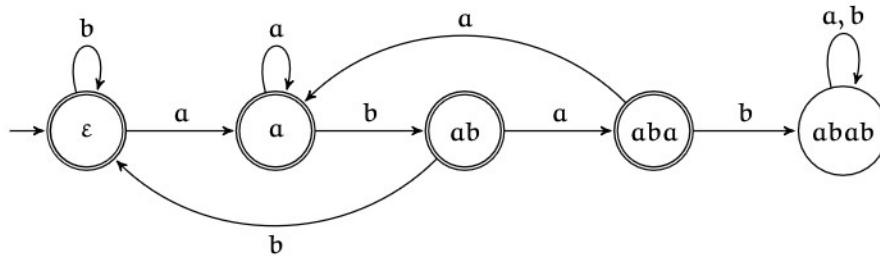
\mathcal{A}_U et \mathcal{A}_\cap sont complets puisque \mathcal{A} et \mathcal{A}' le sont, et pour tout mot $u \in \Sigma^*$, on a :

$$\begin{aligned} u \in \mathcal{L}(\mathcal{A}_U) &\Leftrightarrow (q_I, q'_I).u \in F_U \\ &\Leftrightarrow (q_I.u, q'_I.u) \in F_U && \text{d'après la propriété VI.9} \\ &\Leftrightarrow (q_I.u, q'_I.u) \in (F \times Q') \cup (Q \times F') \\ &\Leftrightarrow q_I.u \in F \text{ ou } q'_I.u \in F' && \text{car les automates sont complets} \\ &\Leftrightarrow u \in \mathcal{L}(\mathcal{A}) \text{ ou } u \in \mathcal{L}(\mathcal{A}') \end{aligned}$$

Donc \mathcal{A}_U reconnaît $L \cup L'$. De même, \mathcal{A}_\cap reconnaît $L \cap L'$.

Correction de l'exercice VI.8 page 6

- Soit L l'ensemble des mots sur $\Sigma = \{a, b\}$ n'admettant pas $abab$ comme facteur. On a $\Sigma^* \setminus L = \mathcal{L}(\Sigma^* abab \Sigma^*)$, donc $\Sigma^* \setminus L$ est rationnel. En admettant le théorème de Kleene, ce langage est également reconnaissable, donc d'après la propriété VI.7 il en est de même pour L . On utilise maintenant l'autre implication du théorème de Kleene pour conclure que L est rationnel.
- Trouver directement une expression régulière décrivant L est délicat, et se convaincre du fait qu'elle convient l'est encore davantage. Le plus raisonnable est sans doute de construire un automate reconnaissant L^C , d'en déduire un automate reconnaissant L puis de déterminer une expression régulière dénotant le langage de cet automate.

FIGURE VI.33 – Un automate reconnaissant L .

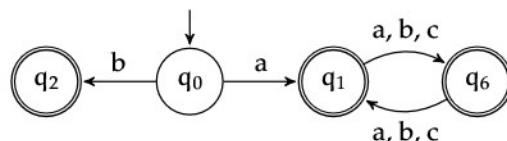
Ensuite, sans chercher à simplifier l'expression :

- on boucle un certain nombre de fois sur l'état ϵ : $(b|a(a|ba^2)^*b^2)^*$;
- on finit par un déplacement de ϵ vers l'un des états acceptants.

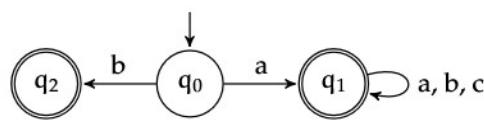
On obtient $(b|a(a|ba^2)^*b^2)^* (\epsilon|a(a|ba^2)^*(\epsilon|b|ba))$.

Correction de l'exercice VI.9 page 7

L'état q_3 n'est pas accessible, les états q_4 et q_5 ne sont pas co-accessibles. On obtient :



Cet automate n'est pas minimal, puisque l'automate suivant lui est équivalent et possède un état de moins :



Correction de l'exercice VI.10 page 9

Figure VI.13 On reconnaît $(a|b|c)^*bab(a|b|c)^*$.

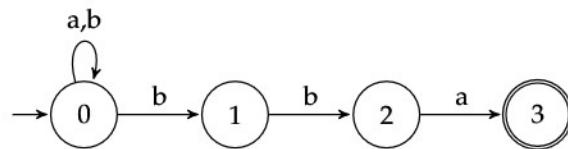
Figure VI.14 On boucle un certain nombre de fois sur l'état 1 par $a|ab^*c$, puis l'on passe sur l'état 2 et l'on y reste par ab^* . On obtient $(a|ab^*c)^* ab^*$.

Figure VI.15 Il faut soit aller à l'état 1 et boucler dessus $(a^+(ab)^*)$, soit aller à l'état 4 et boucler dessus (a^+ba^*) . On obtient donc $a^+(ab)^*|a^+ba^* \equiv a^+((ab)^*|ba^*)$.

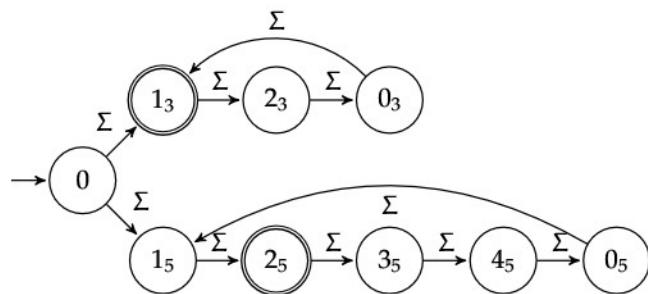
Figure VI.16 On peut aller de q_1 en q_0 en faisant a^+ ou de q_3 en q_0 en faisant b^*b^2 . Il faut ensuite boucler sur q_0 en faisant $(ab)^*$, et le langage reconnu est donc dénoté par $(a^+|b^*b^2)(ab)^*$.

Correction de l'exercice VI.11 page 9

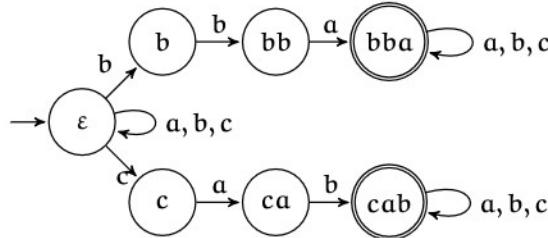
1.



2.



3.



Correction de l'exercice VI.13 page 11

Soit $\mathcal{A} = (Q, I, F, \delta)$ un automate (non nécessairement déterministe) reconnaissant L , on définit $\mathcal{A}' = (Q, I', F', \eta)$ par :

- $I' = F$;
- $F' = I$;
- $\eta(q, a) = \{q' \in Q | q \in \delta(q', a)\}$.

Autrement dit, \mathcal{A}' est obtenu à partir de \mathcal{A} en inversant toutes les flèches et en échangeant les états initiaux et les états acceptants. Pour tous $q, q' \in Q$ et $u \in \Sigma^*$, on a $q' \in \delta^*(q, u)$ si et seulement si $q \in \eta^*(q', \bar{u})$ (par récurrence sur la longueur de u). Ainsi :

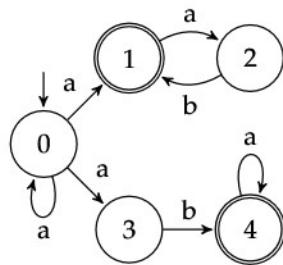
$$\begin{aligned}
 u \in \bar{L} &\Leftrightarrow \bar{u} \in L \\
 &\Leftrightarrow \exists q \in I \exists q' \in F, q' \in \delta^*(q, \bar{u}) \\
 &\Leftrightarrow \exists q \in I \exists q' \in F, q \in \eta^*(q', u) \\
 &\Leftrightarrow \exists q \in F' \exists q' \in I', q \in \eta^*(q', u) \\
 &\Leftrightarrow u \in \mathcal{L}(\mathcal{A}'')
 \end{aligned}$$

On a donc $\bar{L} = \mathcal{L}(\mathcal{A}'')$: \bar{L} est reconnaissable.

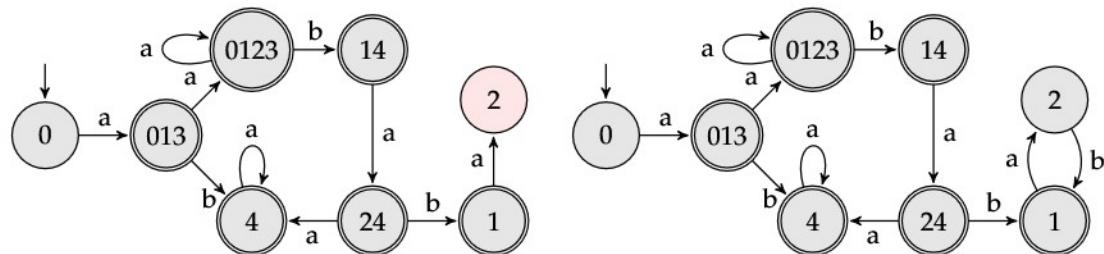
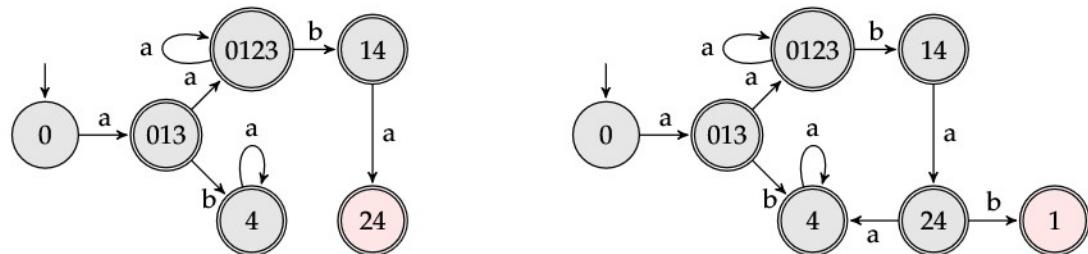
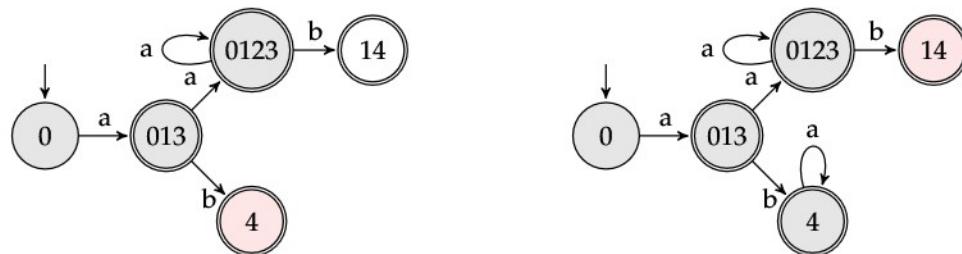
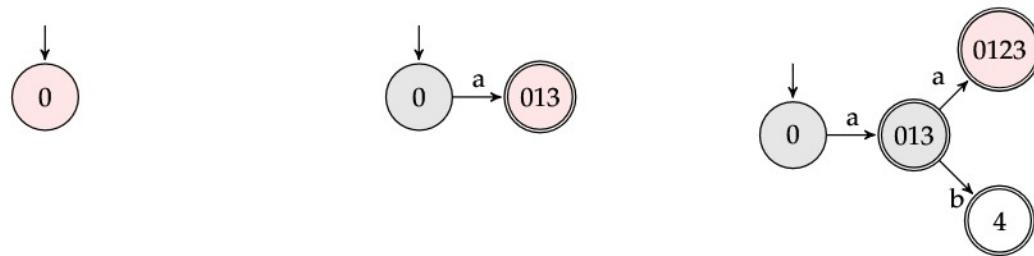
Remarques

- On aurait pu partir d'un automate \mathcal{A} déterministe, mais même dans ce cas l'automate \mathcal{A}' serait *a priori* non déterministe.
- L'automate de la figure VI.16 est la version « miroir » de celui de la figure VI.4. On peut vérifier que les langages reconnus sont bien miroirs l'un de l'autre.

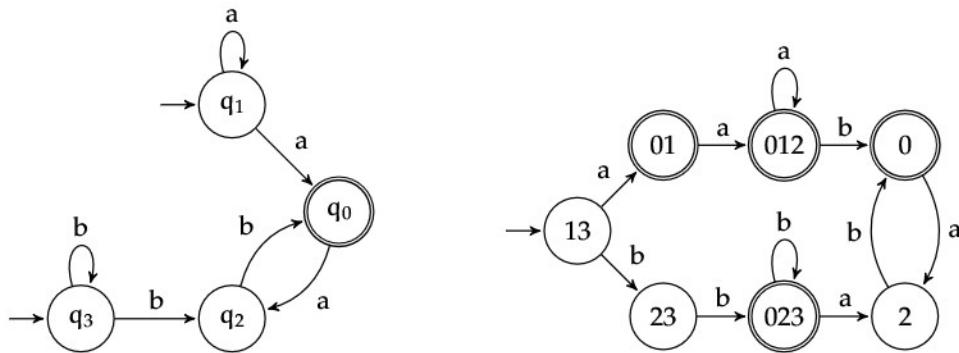
Correction de l'exercice VI.14 page II



Ci-dessous, on a représenté en gris les états déjà traités, en rouge l'état actif et en blanc les états à traiter non encore actifs.



Correction de l'exercice VI.15 page 11

FIGURE VI.34 – L'automate B_3 et son déterminisé.

Correction de l'exercice VI.16 page 13

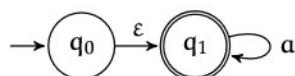
1. On a $E(q_0) = \{q_0, q_1, q_2\}$ et $E(q) = \{q\}$ pour les autres états.
2. ■ $E(q_0) = \{q_0, q_1, q_2, q_4\}$ ■ $E(q_3) = \{q_3\}$ ■ $E(q_6) = \{q_6\}$
■ $E(q_1) = \{q_1, q_2\}$ ■ $E(q_4) = \{q_4\}$
■ $E(q_2) = \{q_2\}$ ■ $E(q_5) = \{q_1, q_2, q_5, q_6\}$

Correction de l'exercice VI.17 page 13

- $\delta^*(q_0, \varepsilon) = E(q_0) = \{q_0, q_1, q_2, q_4\}$
- $\delta^*(q_0, b) = \emptyset$
- $\delta^*(q_0, c) = \{q_5, q_6, q_1, q_2, q_3\}$
- $\delta^*(q_0, bc) = \emptyset$
- $\delta^*(q_0, cb) = \{q_2, q_3\}$

Correction de l'exercice VI.18 page 13

1. Dans l'automate ci-dessous, on a $\delta^*(q_0, a) = \{q_1\}$, mais $\delta(q_0, a) = \emptyset$ donc $E(\delta^*(\delta(q, a), \varepsilon)) = E(\delta^*(\emptyset, \varepsilon)) = E(\emptyset) = \emptyset$: la réponse est non.



On a en revanche $\delta^*(q, aw) = \delta^*(\delta(E(q), a), w)$.

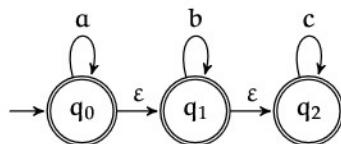
2. Par récurrence sur $|v|$:

- si $v = \varepsilon$, alors $\delta^*(X, u\varepsilon) = \delta^*(X, u)$ et $\delta^*(\delta^*(X, u), \varepsilon) = E(\delta^*(X, u))$. Or $\delta^*(X, u)$ est de la forme $E(Y)$ pour un certain Y (que u soit égal à ε ou non), donc $E(\delta^*(X, u)) = E(E(Y)) = E(Y) = \delta^*(X, u)$ d'après la remarque suivant la définition de E .
- sinon, $v = wa$ avec $a \in \Sigma$. On a alors

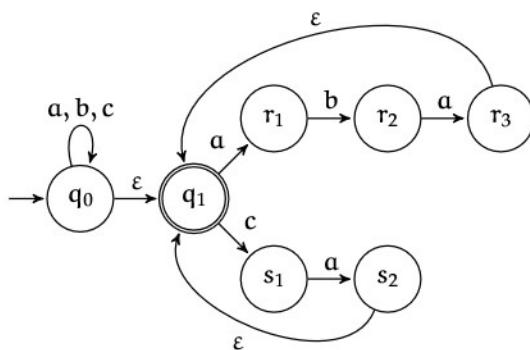
$$\begin{aligned}
\delta^*(q, uw) &= E(\delta(\delta^*(q, uw), a)) && \text{par définition de } \delta^* \\
&= E(\delta(\delta^*(\delta^*(q, u), w), a)) && \text{par hypothèse de récurrence} \\
&= \delta^*(\delta^*(q, u), wa) && \text{par définition de } \delta^*
\end{aligned}$$

Correction de l'exercice VI.19 page 14

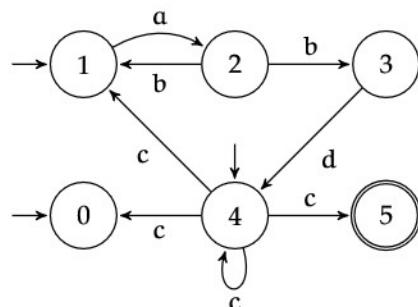
1.



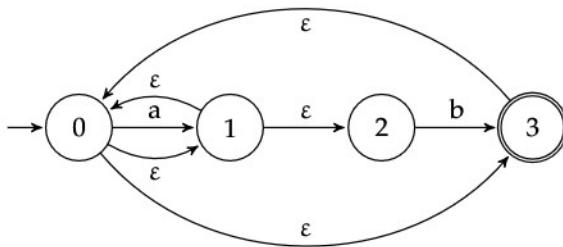
2.



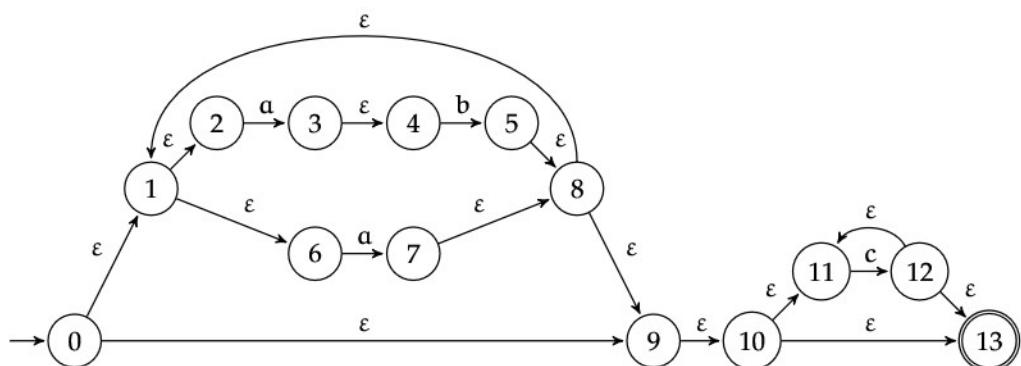
3.

Correction de l'exercice VI.20 page 14**Correction de l'exercice VI.21 page 15**

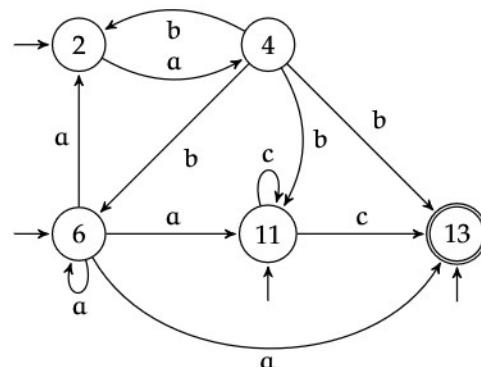
Si $T(f)$ possède un unique état initial, sans transition entrante, et un unique état acceptant, sans transition sortante, alors l'automate proposé reconnaît bien f^* . En revanche, il ne vérifie plus ces conditions : son état initial possède une transition entrante et son état final une transition sortante. En soi, ce n'est pas un problème, mais cela le devient si l'on réapplique ensuite cette construction. Ainsi, pour $(a^*b)^*$, on obtiendrait

**Correction de l'exercice VI.22 page 16**

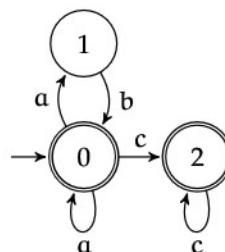
1.



2. On élimine les ϵ -transitions, et on en profite pour supprimer tous les états (non acceptants) pour lesquels il n'y a plus de transition sortante : ils ne peuvent être co-accessibles.



On constate que tous les états sont accessibles : l'automate est donc émondé. Notons que l'on peut facilement trouver un automate non déterministe équivalent possédant moins d'états :



Correction de l'exercice VI.23 page 16

1. $P(L) = \{a\}$, $S(L) = \{b\}$ et $F(L) = \{ab, bb, ba\}$. Le langage n'est pas local car $u = ab$ respecte les contraintes mais n'est pas dans L .
2. $P(L) = \{a\}$, $S(L) = \{a\}$ et $F(L) = \{aa\}$. Le langage est local.
3. $P(L) = \{a\}$, $S(L) = \{b\}$ et $F(L) = \{ab, ba\}$. Le langage est local.
4. $P(L) = \{a, c\}$, $S(L) = \{b, d\}$ et $F(L) = \{ab, ba, bc, cd, dc\}$. Le langage est local.
5. $P(L) = \{a\}$, $S(L) = \{a, b\}$ et $F(L) = \{ab, ba\}$. Le langage n'est pas local car aba respecte les contraintes mais n'est pas dans L .
6. $P(L) = \{a\}$, $S(L) = \{a, b\}$ et $F(L) = \{ab, ba, aa\}$. Le langage n'est pas local car aab respecte les contraintes mais n'est pas dans L .

Correction de l'exercice VI.24 page 17

Soient L_1 et L_2 locaux et $u = u_1 \dots u_n$ avec $n \geq 1$. On note $P_1 = P(L_1)$, $S_1 = S(L_1) \dots$ On a :

$$u \in L_1 \cap L_2 \Leftrightarrow \begin{cases} u_1 \in P_1 \cap P_2 \\ u_n \in S_1 \cap S_2 \\ \forall i \in [1 \dots n-1] \quad u_i u_{i+1} \in F_1 \cap F_2 \end{cases}$$

Donc $(L_1 \cap L_2) \setminus \{\epsilon\} = (P\Sigma^* \cap \Sigma^*S) \setminus \Sigma^*N\Sigma^*$ avec $P = P_1 \cap P_2$, $S = S_1 \cap S_2$ et $N = \Sigma^2 \setminus (F_1 \cap F_2) \subset \Sigma^2$. Donc $L_1 \cap L_2$ est local.

Correction de l'exercice VI.25 page 19

Automate reconnaissant $(a|bc)^*$. On a quatre états étiquetés ϵ , a , b et c . Pour les transitions et les états acceptants, il nous faut P , S et F , que l'on peut calculer « de tête » ici (en étant précautionneux) :

- $P = \{a, b\}$, d'où les transitions (ϵ, a, a) et (ϵ, b, b) ;
- $S = \{a, c\}$, donc a et c sont acceptants;
- $F = \{aa, ab, bc, cb, ca\}$, ce qui donne cinq autres transitions;
- $\epsilon \in \mathcal{L}((a|bc)^*)$, donc ϵ est acceptant.

On obtient :

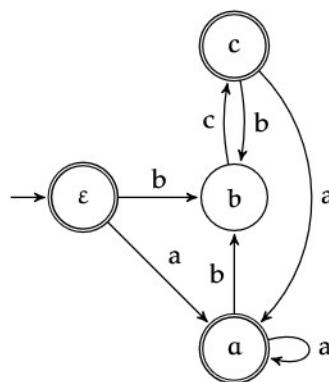
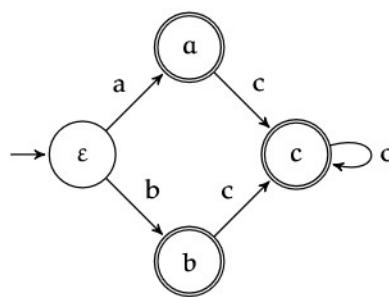


FIGURE VI.35 – Automate local pour $(a|bc)^*$.

Automate reconnaissant $(a|b)c^*$. On procède de même, en notant que cette fois-ci l'état ϵ n'est pas acceptant. On obtient :

FIGURE VI.36 – Automate local pour $(a|b)c^*$.**Correction de l'exercice VI.28 page 24**

Supposons \mathcal{P} rationnel, soient n une longueur de pompage et $u \in \mathcal{P}$ tel que $p = |u| \geq n$. On applique le lemme de l'étoile et l'on obtient $u = xyz$ avec $y = r$ et tous les xy^kz dans \mathcal{P} : ainsi, tous les $p + kr$ sont premiers. En particulier, $p + pr = p(r+1)$ est premier, ce qui est absurde puisque $r \geq 1$ et $p \geq 2$ (p premier).

Correction de l'exercice VI.29 page 24

Supposons L rationnel, et soit alors A un automate à n états le reconnaissant. Posons $u = a^n b^n$:

- $|u|_a = |u|_b = n$ donc $u \in L$;
- d'après le lemme de l'étoile, on a donc $u = xyz$ avec $|xy| \leq n$, $|y| \geq 1$ et $\mathcal{L}(xy^*z) \subset L$;
- comme $|xy| \leq n$, on a en fait $x = a^k$ et $y = a^p$ avec $p \geq 1$;
- comme $\mathcal{L}(xy^*z) \subset L$, on a $xz \in L$;
- cependant, $|xz|_a = n - p < n$ et $|xz|_b = n$: absurde.

Donc L n'est pas rationnel.

Correction de l'exercice VI.30 page 26

Soient $L = \{u \in \Sigma^* \mid u = \bar{u}\}$ (où \bar{u} désigne le miroir de u) et a, b deux lettres distinctes de Σ . Considérons la famille des $L_n = (b^n a)^{-1} L$, $n \in \mathbb{N}$. Les L_n sont deux à deux distincts :

- $b^n \in L_n$ puisque $\overline{b^n a b^n} = b^n a b^n$;
- $b^n \notin L_p$ si $p \neq n$ puisque dans ce cas $\overline{b^p a b^n} = b^n a b^p \neq b^p a b^n$.

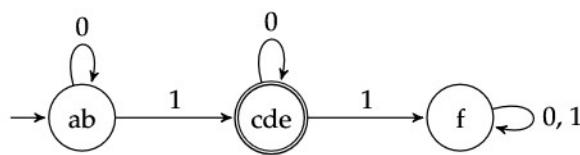
L a donc une infinité de résiduels, ce qui montre qu'il n'est pas rationnel.

Remarque

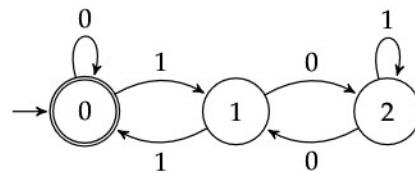
On peut très facilement reformuler cette démonstration de manière à ne pas utiliser la notion de résiduel : on suppose que L est reconnu par un automate déterministe complet A et l'on montre que $q_1.b^n a \neq q_1.b^{n'} a$ si $n \neq n'$, ce qui est absurde car A a un nombre fini d'états.

Correction de l'exercice VI.31 page 26

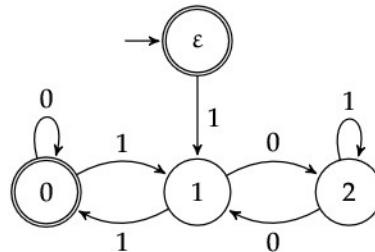
1. On a de manière évidente $L_f = \emptyset$ puis $L_c = L_d = L_e = 0^*$. On en déduit $L_a = (00)^*(10^*|010^*)$ et $L_b = (00)^*(10^*|010^*)$, donc en particulier $L_a = L_b$. On peut remarquer que $L_a = 0^*10^*$.
2. On a donc trois classes d'équivalence : $\{L_a, L_b\}$, $\{L_c, L_d, L_e\}$ et $\{L_f\}$. On obtient alors l'automate minimal :

FIGURE VI.37 – L’automate minimal pour 0^*10^* .**Correction de l’exercice VI.32 page 27**

Si u est la représentation binaire de n , alors u_0 représente $2n$ et u_1 représente $2n + 1$. On en déduit l’automate suivant, où le numéro de l’état correspond à la valeur modulo 3 du mot binaire déjà lu :



Pour se limiter aux représentations canoniques, il faut n’accepter que le mot vide et les mots commençant par un 1 (tout en respectant les contraintes sur la valeur modulo 3). On obtient :

**Correction de l’exercice VI.33 page 27**

Il y a trois boucles « élémentaires » de la forme $0q_1 \dots q_n 0$, où les q_i sont différents de 0 :

- une boucle 010 décrite par ab^+ ;
- une boucle 0120 décrite par ab^*ab^+ ;
- une boucle 01230 décrite par $ab^*ab^*ab^*(a|b)$.

On obtient donc l’expression $(ab^+|ab^*ab^+|ab^*ab^*ab^*(a|b))^*$, que l’on peut éventuellement factoriser en $\left(ab^*(b|ab^*(b|ab^*(a|b)))\right)^*$.

Correction de l’exercice VI.34 page 27

On considère un automate déterministe émondé $A = (\Sigma, Q, q_1, F, \delta)$ reconnaissant L .

- $\text{pref}(L)$ est reconnu par l’automate obtenu en rendant acceptants tous les états de A .
- $\text{suff}(L)$ est reconnu par l’automate obtenu en rendant initiaux tous les états de A .
- $\text{fact}(L)$ est reconnu par l’automate obtenu en rendant initiaux et acceptants tous les états de A .

Correction de l'exercice VI.36 page 27

Notons P' , S' , F' l'ensemble des premières lettres, dernières lettres et facteurs de taille 2 de $\text{fact}(L)$ (et P , S , F les mêmes ensembles pour L). De manière immédiate, on a :

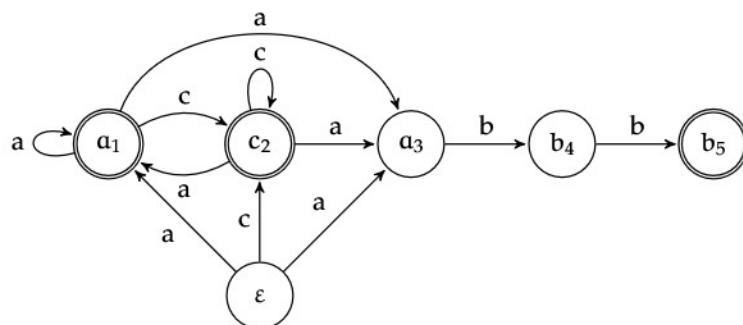
- P' et S' qui sont égaux à l'ensemble des lettres apparaissant dans les mots de L ;
- $F' = F$.

Il reste à vérifier qu'un mot non vide respectant ces contraintes locales appartient bien à $\text{fact}(L)$; soit u un tel mot.

- Si u est de longueur 1, alors u est réduit à une lettre de P' , qui, par définition, apparaît dans au moins un mot de L , et donc $u \in \text{fact}(L)$.
- Sinon, écrivons $u = xvy$ avec $x, y \in \Sigma$ et $v \in \Sigma^*$. Comme $x \in P'$, il existe un mot $\alpha x \beta \in L$ (avec $\alpha, \beta \in \Sigma^*$), et de même il existe $\gamma y \delta \in L$. Considérons alors le mot $w = \alpha x v y \beta$:
 - il admet $u = xvy$ comme facteur ;
 - sa première lettre est celle de α , donc appartient à P ;
 - de même, sa dernière lettre appartient à S ;
 - ses facteurs de taille 2 sont soit des facteurs de αx , soit des facteurs de $x v y$, soit des facteurs de $y \beta$. Dans les trois cas, ces facteurs appartiennent à $F = F'$.

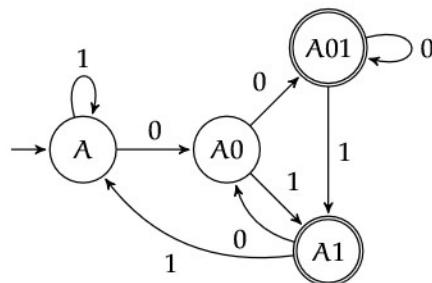
Le mot w vérifie donc les contraintes locales de L : comme L est local, cela implique que $w \in L$ et donc que $u \in \text{fact}(L)$.

Ainsi, $\text{fact}(L)$ est bien local.

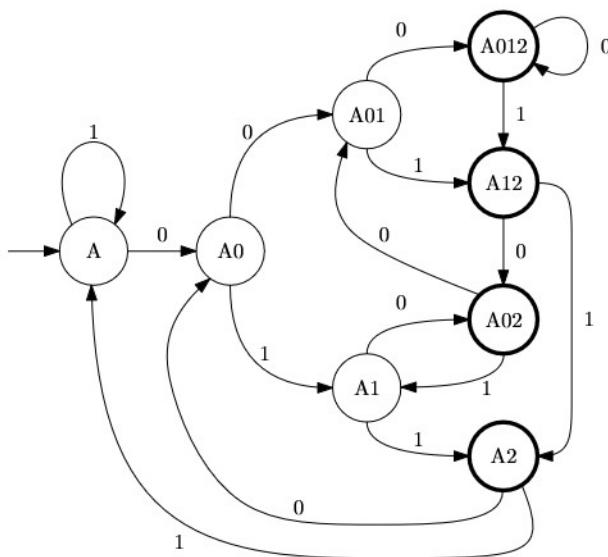
Correction de l'exercice VI.37 page 27**Correction de l'exercice VI.40 page 28**

1. L'automate reconnaît $L_n = (0|1)^*0(0|1)^{n-1}$.

2. On obtient :



3. On voit apparaître un arbre (surtout si l'on ignore les transitions partant des « feuilles », c'est-à-dire des états acceptants) :



4. L'automate initial possède $n + 1$ états, on peut donc *a priori* avoir jusqu'à 2^{n+1} états dans le déterminisé. Cependant, tous les états de l'automate déterminisé contiennent A , et il y en a donc au plus 2^n . Le fait qu'on ait *exactement* 2^n états « se voit » : ce n'est bien sûr pas une démonstration, mais la question suivante en fournit une.

5. Soit A un automate déterministe reconnaissant L_n et q_1 son état initial. Soient $u, v \in \Sigma^n$ avec $u \neq v$ (on pose $\Sigma = \{0, 1\}$) ; u et v diffèrent au moins par une lettre, et l'on peut donc supposer sans perte de généralité que $u_i = 0$ et $v_i = 1$ pour un certain $i \in [0 \dots n - 1]$.

On peut donc écrire $u = a0b$ avec $|b| = n - 1 - i$ et $v = c1d$ avec $|d| = n - 1 - i$. On a

alors $u1^i = a0 \overbrace{b1^i}^{\in L} \in L$ et $v1^i = c1 \overbrace{d1^i}^{\notin L} \notin L$, ce qui implique $q_1.u1^i \neq q_1.v1^i$ et donc $q_1.u \neq q_1.v$. On en déduit que l'application $\varphi : \Sigma^n \rightarrow Q | w \mapsto q_1.w$ est injective, et donc que $|Q| \geq 2^n$. D'après la question précédente, l'automate obtenu en déterminisant A_n est donc minimal.

Remarque

On a ici un exemple où la déterminisation est très proche du pire cas (un AFND à $n + 1$ états donne un AFD à 2^n états après déterminisation) et où l'automate obtenu est minimal. Cela montre que, même si le non-déterminisme ne permet pas de reconnaître plus de langages, il permet dans certains cas de reconnaître un langage en utilisant un nombre exponentiellement plus petit d'états qu'en se limitant aux automates déterministes.

Correction de l'exercice VI.41 page 28

1. On montre par induction structurelle sur e que $n \leq 2|e|$.
 - Dans les cas $e = \epsilon$ et $e = a \in \Sigma$, on a $\text{taille}(e) = 1$ et $n = 2$.
 - Dans le cas $e = f^*$, on a $\text{taille}(e) = \text{taille}(f) + 1$ et $n_e = n_f + 2$, donc $n_e \leq 2|f| + 2 = 2|e|$ d'après l'hypothèse d'induction.
 - Dans le cas $e = f + g$, on a $\text{taille}(e) = \text{taille}(f) + \text{taille}(g) + 1$ et $n_e = n_f + n_g + 2$, donc à nouveau $n_e \leq 2\text{taille}(f) + 2\text{taille}(g) + 2 = 2\text{taille}(e)$ d'après l'hypothèse d'induction.
 - De même pour $e = fg$, sauf que l'inégalité est stricte ici puisque $n_e = n_f + n_g$.
2. Les états sont clairement tous accessibles et co-accessibles (puisque n'a pas de \emptyset dans l'expression initiale).
3. Les états non acceptants n'ayant pas de transition sortante étiquetée par autre chose que ϵ ne sont plus co-accessibles, et ce sont les seuls. Tous les états restent accessibles (c'est vrai de manière générale : notre méthode pour l'élimination des ϵ -transitions ne rend jamais un état inaccessible).

4. Chaque occurrence d'une lettre dans l'expression crée exactement un état avec une transition sortante non spontanée, et ce sont les seuls. Dans l'automate émondé, on a donc un état par occurrence d'une lettre, plus l'unique état acceptant de l'automate (qui n'a pas de transition sortante). Finalement, on obtient $|e| + 1$ états dans l'automate émondé, exactement comme pour l'automate de Glushkov.

Correction de l'exercice VI.42 page 28

1. Supposons L' rationnel. Les langages rationnels étant stables par intersection (*via* la stabilité des langages reconnaissables et le théorème de Kleene), on aurait alors $L \cap L'$ rationnel. Donc L' n'est pas rationnel : la proposition est vraie..
2. Soit L' un langage non rationnel, par exemple $L' = \{a^n b^n \mid n \geq 0\}$ et $L = \emptyset$, qui est rationnel. On a $L \cap L' = \emptyset$ rationnel : la proposition est fausse.
3. Les langages rationnels sont stables par étoile de Kleene, par construction.
4. Soit $L = \{a^{n^2} \mid n \geq 0\}$, qui n'est pas rationnel comme nous l'avons vu (n'importe quel langage non rationnel contenant a ferait l'affaire). On a $L^* = a^*$, qui est rationnel : la proposition est fausse.
5. On a $L \setminus L' = L \cap L'^c$ (en notant L'^c le complémentaire $\Sigma^* \setminus L'$ de L' dans Σ^*). Or les langages rationnels sont stables par intersection et passage au complémentaire, donc $L \setminus L'$ est rationnel : la proposition est vraie.
6. Prenons $\Sigma = \{a, b\}$, $L = \{u \in \Sigma^* \mid |u|_a = |u|_b\}$ et $L' = L'^c \cup \{\epsilon\} = \{u \in \Sigma^* \mid |u|_a \neq |u|_b\} \cup \{\epsilon\}$. Nous avons vu que L n'était pas rationnel. Si L' était rationnel, alors $L' \cap \Sigma^+ = L^c$ le serait également, ce qui n'est pas le cas (stabilité des langages rationnels par passage au complémentaire). Donc ni L ni L' ne sont rationnels. Mais il est clair que $LL' = \Sigma^*$, qui est bien sûr rationnel : la propriété est donc fausse.

Correction de l'exercice VI.43 page 29

1. On construit l'automate de Glushkov associé à l'expression (ou l'automate de Thompson) et on l'exécute sur le mot. On peut déterminiser l'automate, mais ce n'est pas nécessaire, et pas rentable si l'on ne veut décider l'appartenance que pour un seul mot.
2. On commence par émonder l'automate, puis l'on détermine si le graphe sous-jacent possède un cycle (algorithme vu en première année, à base de parcours en profondeur). Si l'automate initial possède des ϵ -transitions, il faut commencer par les éliminer.
3. Il est inutile de construire un automate, on peut répondre très simplement en parcourant l'arbre de l'expression régulière : se référer à l'exercice ??.
4. À nouveau, inutile de construire un automate, on peut travailler directement sur l'arbre de l'expression : se référer à l'exercice ??.
5. On effectue un parcours de graphe à partir de chacun des états initiaux, et l'on regarde si l'on atteint au moins un état acceptant.
6. On construit l'automate de Glushkov associé à l'expression et on le déterminise : on obtient un automate (déterministe) complet reconnaissant $\mathcal{L}(e)$. En inversant états acceptants et non acceptants, on obtient un automate pour $\mathcal{L}(e)^c$, ce qui permet d'après la question précédente de tester la vacuité de $\mathcal{L}(e)$.
7. On commence par construire deux automates déterministes complets A et B reconnaissant $\mathcal{L}(e)$ et $\mathcal{L}(f)$ (Glushkov puis déterminisation). On peut alors construire A' reconnaissant $\mathcal{L}(e)^c$ et B' reconnaissant $\mathcal{L}(f)^c$ comme à la question précédente. On construit ensuite deux automates produit reconnaissant respectivement $\mathcal{L}(e) \cap \mathcal{L}(f)^c$ et $\mathcal{L}(e)^c \cap \mathcal{L}(f)$. Les deux langages sont égaux si et seulement si ces deux automates produits reconnaissent le langage vide.

Correction de l'exercice VI.44 page 29

Il faut alors prendre $I' = I$ et $F' = \{q \in Q \mid E(q) \cap F \neq \emptyset\}$. On montre alors par récurrence sur $|u|$ que $\delta^*(I, u) = E(\eta^*(I, u))$.

Correction de l'exercice VI.45 page 29

1. a. On a immédiatement $n\mathcal{R}^+p \iff n < p$ et $n\mathcal{R}^*p \iff n \leq p$.
 - b. On prend $n\mathcal{R}p \Leftrightarrow (p = 2n \text{ ou } p = 2n+1)$ (qui vérifie bien la condition sur le cardinal), et l'on montre par récurrence sur $k \geq 1$ que $n\mathcal{R}^k p \Leftrightarrow \exists r < 2^k, p = 2^k n + r$.
 - Pour $k = 1$, c'est précisément la définition choisie.
 - Si $n\mathcal{R}^{k+1}p$, alors il existe q tel que $n\mathcal{R}^k q$ et $q\mathcal{R}p$. L'hypothèse de récurrence nous donne $q = 2^k n + r$ avec $r < 2^k$, on distingue ensuite les deux cas :
 - si $p = 2q$, alors $p = 2^{k+1}n + 2r$ avec $2r < 2^{k+1}$;
 - si $p = 2q + 1$, alors $p = 2^{k+1}n + 2r + 1$, et comme $r \leq 2^k - 1$, on a bien $2r + 1 < 2^{k+1}$.
 - Si $p = 2^{k+1}n + r$ avec $r < 2^{k+1}$, alors on distingue deux cas :
 - si $r < 2^k$, alors on prend $q = 2n$ et $r' = r$;
 - si $2^k \leq r < 2^{k+1}$, alors on prend $q = 2n + 1$ et $r' = r - 2^k$.
- Dans les deux cas, on a bien $n\mathcal{R}q$ et $q\mathcal{R}^kp$, puisque $p = 2^k q + r'$ avec $0 \leq r' < 2^k$.
2. Une récurrence immédiate sur p montre que si $x\mathcal{R}^ny$ et $y\mathcal{R}^pz$, alors $x\mathcal{R}^{n+p}z$. On en déduit que \mathcal{R}^+ est bien une relation transitive.
Pour montrer que c'est la plus petite relation transitive contenant \mathcal{R} , on considère une relation transitive \mathcal{P} contenant \mathcal{R} et l'on montre par récurrence sur k que \mathcal{P} contient \mathcal{R}^k pour tout $k \geq 1$. À nouveau, c'est immédiat.
 3. C'est le graphe obtenu en remplaçant chaque composant connexe de G par un graphe complet.
 4. a. Il faut et il suffit que le graphe soit acyclique : cela correspond exactement au fait que \mathcal{R}^* soit antisymétrique (elle est toujours transitive et réflexive).
 - b. On exige qu'il y ait toujours un chemin de x à y ou un chemin de y à x , c'est-à-dire que le graphe soit semi-connexe.
 - c. On a \mathcal{R}^* inclus dans \preceq .

Correction de l'exercice VI.46 page 30

1. Pour $u \in \Sigma^*$, on a :

$$\begin{aligned} u \in \mathcal{L}(A'_q) &\Leftrightarrow (q_0, q).u = (q, q_f) \\ &\Leftrightarrow q_0.u = q \text{ et } q.u = q_f \\ &\Leftrightarrow q_0.u = q \text{ et } q_0.uu = q_f \\ &\Leftrightarrow q_0.u = q \text{ et } uu \in \mathcal{L}(A) \\ &\Leftrightarrow q_0.u = q \text{ et } u \in \sqrt{\mathcal{L}(A)} \end{aligned}$$

Autrement dit, $\mathcal{L}(A'_q)$ est constitué des mots de $\sqrt{\mathcal{L}(A)}$ tels que $q_0.u = q$.

2. Soit L rationnel et A déterministe complet reconnaissant A . Par disjonction de cas sur l'état $q_0.u$, on a d'après la question précédente $u \in \sqrt{\mathcal{L}(A)} \Leftrightarrow \exists q \in Q, u \in \mathcal{L}(A'_q)$. Ainsi, $L = \sqrt{\mathcal{L}(A)} = \bigcup_{q \in Q} \mathcal{L}(A'_q)$. Or chacun des $\mathcal{L}(A'_q)$ est rationnel et les langages rationnels sont stables par union finie, donc \sqrt{L} est rationnel.

Correction de l'exercice VI.49 page 31

Soit $A = (\Sigma, Q, q_1, F, \delta)$ un automate déterministe complet reconnaissant L . On considère l'automate (non déterministe) A' obtenu à partir de A en prenant comme états initiaux l'ensemble $I = \{q \in Q \mid \exists u \in K, q_1.u = q\}$ (le reste de l'automate est inchangé). Cet automate reconnaît $K^{-1}L$:

$$\begin{aligned} v \in \mathcal{L}(A') &\Leftrightarrow \exists q \in I, q.v \in F \\ &\Leftrightarrow \exists q \in Q \exists u \in K, q_1.u = q \text{ et } q.v \in F \\ &\Leftrightarrow \exists u \in K, q_1.uv \in F \\ &\Leftrightarrow \exists u \in K, uv \in \mathcal{L}(A) = L \end{aligned}$$

Correction de l'exercice VI.50 page 31

Supposons qu'il existe une infinité de nombres de Mersenne, et que L soit rationnel. Soit alors n une longueur de pompage pour L : il existe nécessairement un $p \geq n$ tel que $2^p - 1$ soit premier (on a aussi $p \geq 2$). Comme $2^p - 1$ s'écrit $u = 1^p$ en binaire, on peut décomposer u en xyz avec $|y| = r \geq 1$ et $xy^*z \subset L$. Ainsi, tous les $2^{p+rk} - 1$ avec $k \geq 0$ sont premiers : pour $k = p$, on obtient $2^{p(r+1)} - 1$ premier. C'est absurde, puisqu'on peut factoriser :

$$2^{p(r+1)} - 1 = (2^p - 1)(1 + 2^p + 2^{2p} + \dots + 2^{rp}).$$

Les deux facteurs sont bien supérieurs ou égaux à 2, on a la contradiction recherchée.

Correction de l'exercice VI.51 page 31

Supposons L rationnel : on a alors $L' = L \cap \mathcal{L}(ab^*c^*)$ rationnel (les langages rationnels sont stables par intersection), or $L' = \{ab^r c^r \mid r \geq 0\}$. C'est absurde (on adapte facilement la preuve du fait que $\{a^r b^r \mid r \geq 0\}$ n'est pas rationnel), donc L n'est pas rationnel.

Montrons que L vérifie le lemme de l'étoile, en prenant 3 comme longueur de pompage. On considère donc un mot $u \in L$ tel que $|u| \geq 3$.

- Si $u = ab^r c^r$ pour un certain $r \geq 1$, alors $u = xyz$ avec $x = \epsilon$, $y = a$ et $z = b^r c^r$. On a bien $|xy| = 1 \leq 3$, et tous les $xy^k z$ sont dans L : pour $k = 1$ dans le premier ensemble, pour les autres k dans le deuxième.
- Si on a $u = b^s c^t$ avec $s + t \geq 1$, alors on prend encore $x = \epsilon$ et $y = b$ ou $y = c$ (suivant si $s > 0$ ou non). À nouveau, la conclusion du lemme de l'étoile est vérifiée.
- Si l'on a $u = aab^s c^t$, on procède de même sauf que l'on pose $x = aa$. On a bien $|xy| = 3$ et tous les $xy^k z$ sont dans L .
- Sinon, on a nécessairement $u = a^r b^s c^t$ avec $r \geq 3$. On prend $x = \epsilon$, $y = a$ et $z = a^{r-1} b^s c^t$. Comme $r - 1 \geq 2$, tous les $xy^k z$ sont dans L (et $|xy| = 1 \leq 3$).

Donc L vérifie le lemme de l'étoile.
