Université de Versailles – Saint-Quentin-en-Yvelines 2020 – 2021
Licence 2 Informatique

IN406 - Théorie des langages – TD

Version du 18 mars 2021

Xavier Badin de Montjoye – xavier.badin-de-montjoye2@uvsq.fr
Pierre Coucheney – pierre.coucheney@uvsq.fr
Franck Quessette – franck.quessette@uvsq.fr
Yann Strozecki – yann.strozecki@uvsq.fr
Sandrine Vial – sandrine.vial@uvsq.fr

Table des matières

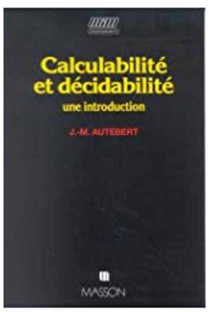
0 Bibliographie

Sur le web

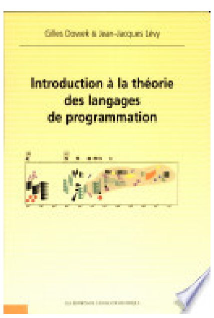
- J.-P. Jouannaud (LiX, Polytechnique) : http ://www.lix.polytechnique.fr/~jouannaud/articles/cours-info-theo.pdf
- F. Yvon (LISMSI, PAris-Saclay), A. Demaille (LRDE, EPITA) : https ://perso.limsi.fr/yvon/classes/thl/thl-2.pdf
- A. Nasr (LIS, Marseille) : http ://pageperso.lif.univ-mrs.fr/ alexis.nasr/Ens/THL/THL.html
- A. Muscholl(LaBRI, Bordeaux) : https ://www.labri.fr/perso/anca/Langages/cours/automates.pdf

Livres

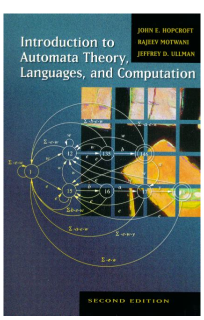
Titre : **Théorie des langages et des automates**
Année : 1994
Auteur : Autebert, Jean-Michel
Éditeur : Masson
ISBN : 2-225-84001-6



Titre : **Calculabilité et décidabilité : une introduction**
Année : 1992
Auteur : Autebert, Jean-Michel
Éditeur : Masson
ISBN : 2-225-82632-3



Titre : **Introduction à la théorie des langages de programmation**
Année : 2006
Auteur : Dowek, Gilles ; Lévy, Jean-Jacques
Éditeur : Les Éditions de lÉcole polytechnique
ISBN : 978-2-7302-1333-2, 2-7302-1333-3



Titre : **Introduction to automata theory, languages, and computation, 3rd Edition**
Année : 2006
Auteur : Hopcroft, John E. ; Motwani, Rajeev ; Ullman, Jeffrey D
Éditeur : Pearson Academic
ISBN : 0-321-45536-3
Le livre en pdf
Page d'Ullman (un des auteurs) avec la correction d'exercices du livre

1 TD 1 – Mot, langage, automate (avec corrigés)

Exercice 1.1 *Alphabet et langage*
Soit $\Sigma = \{a, b\}$ un alphabet, comment peut-on définir les langages Σ^0, Σ^1, Σ^2 ? Comment définir Σ^* ?

Correction

D'après les définitions du cours, pour que la notion d'exposant fasse sens, il faut considérer $\{a, b\}$ comme une lettre, un mot ou alors un langage. Les deux premières options étant évidemment impossible puisque nous considérons un ensemble, on interprète donc Σ comme un langage. Ainsi, on ne fait pas la distinction entre l'ensemble de lettre et le langage composé des mots d'une seule lettre.

À partir de là, on revient au définition du cours pour établir que $\Sigma^0 = \{\epsilon\}$, Σ^1 est le langage $\{a, b\}$ et $\Sigma^2 = \{aa, ab, ba, bb\}$. Autrement dit, on remarque que pour tout entier k, Σ^k correspond au language des mots de taille k sur l'alphabet Σ.

De plus on a par définition

$$\Sigma^* = \bigcup_{n=0}^{n=+\infty} \Sigma^k$$

Ainsi, Σ^* correspond au langage des mots de tailles finis sur l'alphabet Σ, donc Σ^* est l'ensemble des mots pouvant être écrits avec les lettres a et b.

Exercice 1.2 *Longueur*
Pour tout mot w sur l'alphabet Σ, $|w|$ est la longueur (nombre de lettres du mot). La longueur est définie par :

— $|\varepsilon| = 0$;
— $\forall a \in \Sigma$, $\forall w$ mot sur Σ, $|aw| = |wa| = 1 + |w|$;

Pour tout alphabet Σ, démontrez que :

$$\forall w_1, w_2 \in \Sigma^*, |w_1 w_2| = |w_1| + |w_2|$$

Correction

On rappelle qu'un mot sur Σ est défini comme la concaténation finie de lettres de Σ. Ainsi, tout mot w est soit vide, soit s'écrit comme $w'a$ avec $a \in \Sigma$ et w' un mot sur Σ composé de moins de lettres que w.

Soit w_1 un mot définie sur Σ. On veut montrer par induction que pour tout mot $w_2 \in \Sigma$, on satisfait la propriété :

$$|w_1 w_2| = |w_1| + |w_2|$$

Initialisation : Pour w_2 le mot vide, on a $w_1 w_2 = w_1 \epsilon = w_1$. Donc

$$|w_1 w_2| = |w_1| = |w_1| + 0 = |w_1| + |w_2|$$

Hérédité : On suppose la propriété vraie pour un mot w_2. Soit $a \in \Sigma$, on va prouver que la propriété reste vraie pour $w_2' = w_2 a$.

On a $|w_1 w_2'| = |w_1 w_2 a|$. Or par définition $|w_1 w_2 a| = 1 + |w_1 w_2|$. On applique l'hypothèse de récurrence sur $|w_1 w_2|$ pour obtenir :

$$|w_1 w_2'| = 1 + |w_1| + |w_2|$$

Or, on sait que $|w_2'| = 1 + |w_2|$. Il en résulte que $|w_1 w_2'| = |w_1| + |w_2'|$.

Conclusion : Cela prouve notre propriété sur w_2' et achève notre induction. L'axiome de récurrence nous permet de conclure que pour tout mot w_2 sur Σ, on a $|w_1 w_2| = |w_1| + |w_2|$.

Cela ayant été montré pour tout mot w_1, on en conclut donc :

$$\forall w_1, w_2 \in \Sigma^*, |w_1 w_2| = |w_1| + |w_2|$$

Exercice 1.3 *Concaténation*

1. Quelle est la concaténation des mots $w_1 = abc$ et $w_2 = cba$?

Correction

On a $w_1 w_2 = abccba$.

2. La concaténation est-elle associative ?

Correction

Pour prouver l'associativité de la concaténation, il nous faut montrer que pour trois mots x, y, z dans Σ^*, on a $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. On note $w = (x \cdot y) \cdot z$ et $w' = x \cdot (y \cdot z)$. Pour un mot m on note m_i la i-ème lettre de m. Si deux mots m et m' ont même longueur et vérifie pour tout $1 \leq i \leq |m|$, $m_i = m'_i$, alors ces mots sont les mêmes.

Ici, $|w| = |xy| + |z| = |x| + |y| + |z| = |x| + |yz| = |w'|$. De plus, sous réserve d'existence, pour $1 \leq i \leq x$, on a $w_i = w'_i = x_i$. Pour $1 \leq i \leq y$, on a $w_{|x|+i} = w'_{|x|+i} = y_i$ et pour $1 \leq i \leq z$, on a $w_{|x|+|y|+i} = w'_{|x|+|y|+i} = z_i$.

Ainsi, $w = w'$ et la concaténation est bien associative.

3. La concaténation est-elle commutative ?

Correction

On note que $w_2 w_1 = cbaabc \neq w_1 w_2$. Cela prouve que la concaténation n'est pas commutative.

4. Donnez deux mots différents w_1 et w_2 de longueur non nulle tels que $w_1 w_2 = w_2 w_1$.

Correction

On peut prendre par exemple $w_1 = a$ et $w_2 = aa$. On a alors bien $w_1 w_2 = w_2 w_1 = aaa$.

Exercice 1.4 *Lemme de Levi*

Lemme de Lévi

Soient w_1, w_2, w'_1 et w'_2 des mots sur un alphabet Σ avec $|w_1| \geq |w'_1|$. Si $w_1 w_2 = w'_1 w'_2$, alors il existe un mot z tel que $w_1 = w'_1 z$ et $z w_2 = w'_2$.

Lemme de Lévi (version 2)

Si deux mots u et v sont des préfixes d'un même mot, alors l'un deux est préfixe de l'autre.

Démontrez le lemme de Levi.

Correction

Soit $w = w_1 w_2 = w'_1 w'_2 = a_1 a_2 \ldots a_n$ avec les $a_i \in \Sigma$.

On pose $l = |w_1|$ et $l' = |w'_1|$. On suppose que $l \geq l'$.

On a $w_1 = a_1 \ldots a_l$ et $w_2 = a_{l+1} \ldots a_n$. De plus on a $w_1 = a_1 \ldots a_{l'}$ et $w'_2 = a_{l'+1} \ldots a_n$. Si on pose $z = a_{l'+1} \ldots a_l$, on note que $w'_1 z = w_1$ et $z w_2 = w'_2$. Ce qui prouve le lemme de Levi.

Exercice 1.5 *Lemme de Levi (application)*

En utilisant le lemme de Levi, montrez que deux mots w_1 et w_2 commutent, c'est à dire que $w_1 w_2 = w_2 w_1$, si et seulement si ils sont puissance d'un même mot.

Correction

Montrons que tous mots m_1, m_2 tel que $m_1 m_2 = m_2 m_1$ sont puissance d'un même mot. Pour ce faire on procède par induction sur la taille de $m_1 m_2$. Soit $\mathcal{H}(n)$ la propriété : "Pour tout mot $m = m_1 m_2$ tel que $|m| \leq n$ et tel que $m_1 m_2 = m_2 m_1$, m_1 et m_2 sont puissance d'un même mot."

Si $|m| = 0$ on a $m_1 = m_2 = \epsilon$. Donc $\mathcal{H}(0)$ est vraie.

On suppose $\mathcal{H}(n)$ vraie pour $n \geq 0$. Montrons que $\mathcal{H}(n+1)$ est aussi vraie. Soit m un mot. Si $|m| < n+1$ alors par $\mathcal{H}(n)$, on a pour tout m_2 qui commute avec m_1, qu'ils sont puissance d'un même mot. Si maintenant, $|m| = n+1$. Soit m_1, m_2 tel que $m = m_1 m_2 = m_2 m_1$. Par symétrie, on peut supposer que

$|m_1| \leq |m_2|$. Par le lemme de Levy, m_1 est donc préfixe de m_2 et il existe x tel que $m_1x = m_2$. On a donc :

$$\boxed{m_1m_1x = m_1xm_1}$$

Ce qui nous donne donc $m_1x = xm_1$. De plus, si $|m_1| = 0$ on a $m_1 = \epsilon = m_2^0$. Si $|m_1| > 0$, $|xm_1| < |m|$ et par hypothèse de récurrence, on a donc un mot y et deux entiers r, s tel que $x = y^r$ et $m_1 = y^s$. Ainsi, $m_2 = y^{r+s}$ et m_1 et m_2 sont puissance d'un même mot. Ce qui prouve $\mathcal{H}(n+1)$ et achève la récurrence.

Exercice 1.6

Soit un alphabet Σ. Soient a et b deux lettres de Σ et w un mot de Σ^*. Si $aw = wb$, que peut-on déduire sur, a, b et w ?

Correction

Soit i le rang de la première lettre de w qui n'est pas égale à a. Comme $aw = wb$, cela implique que la $i-1$-ème lettre de w doit être différente de a et si $i = 1$, cela impliquerait que a ne vaut pas a ce qui n'a pas de sens. Ainsi, w ne contient pas de lettres autres que a. Cela implique que la lettre b et la lettre a sont les mêmes.

Ainsi, $a = b$ et il existe k tel que $w = a^k$. On remarque que si ces conditions sont vérifiées on a bien $aw = wb$.

Exercice 1.7 *Premiers automates*

Cet exercice peut-être fait en ligne sur le site Automata Tutor v3 : https://automata-tutor.model.in.tum.de/index. Vous devez d'abord vous inscrire sur le site : bouton "Register" en haut à droite, puis vous connecter bouton "Login" en haut à droite, enfin dans la section "Enroll in course", dans le champ "Course ID :" il faut mettre `IN406` et dans le champ "Course Password :", il faut mettre `5355ZUUH`.

Soit $\Sigma = \{a, b, c\}$ un alphabet. Essayez de construire les automates reconnaissant les langages suivants :

— L_1, l'ensemble des mots qui commencent par a ;

Correction

$\mathcal{A}_1 = (\Sigma, \{q_0, q_1\}, q_0, \{q_1\}, T_1)$ avec $T_1 = \{(q_0, a, q_1), (q_1, a, q_1), (q_1, b, q_1), (q_1, c, q_1)\}$.

— L_2, l'ensemble des mots qui ne contiennent pas de c ;

Correction

$\mathcal{A}_2 = (\Sigma, \{q_0\}, q_0, \{q_0\}, T_2)$ avec $T_2 = \{(q_0, a, q_0), (q_0, b, q_0)\}$.

— L_3, l'ensemble des mots qui contiennent au moins un a ;

Correction

$\mathcal{A}_3 = (\Sigma, \{q_0, q_1\}, q_0, \{q_1\}, T_3)$ avec $T_3 = \{(q_0, a, q_1), (q_0, b, q_0), (q_0, c, q_0), (q_1, a, q_1), (q_1, b, q_1), (q_1, c, q_1)\}$.

— L_4, l'ensemble des mots qui contiennent au plus un a ;

Correction

$\mathcal{A}_4 = (\Sigma, \{q_0, q_1\}, q_0, \{q_0, q_1\}, T_4)$ avec $T_4 = \{(q_0, a, q_1), (q_0, b, q_0), (q_0, c, q_0), (q_1, b, q_1), (q_1, c, q_1)\}$

— $L_{51} = \Sigma^*$, $L_{52} = \emptyset$, $L_{53} = \{\epsilon\}$;

Correction

$\mathcal{A}_{51} = (\Sigma, \{q_0\}, q_0, \{q_0\}, T_{51})$ avec $T_{51} = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, c, q_0)\}$
$\mathcal{A}_{52} = (\Sigma, \{q_0\}, q_0, \emptyset, \emptyset)$
$\mathcal{A}_{53} = (\Sigma, \{q_0\}, q_0, \{q_0\}, \emptyset)$

— L_6, l'ensemble des mots qui ont un nombre pair de a (nommez les états de façon pertinente) ;

Correction

$\mathcal{A}_6 = (\Sigma, \{a_p, a_i\}, a_p, \{a_p\}, T_6)$ avec $T_6 = \{(a_p, a, a_i), (a_p, b, a_p), (a_p, c, a_p), (a_i, a, a_p), (a_i, b, a_i), (a_i, c, a_i)\}$

— L_7, l'ensemble des mots qui ont un nombre impair de b (nommez les états de façon pertinente) ;

Correction

$\mathcal{A}_7 = (\Sigma, \{b_p, b_i\}, b_p, \{b_i\}, T_7)$ avec $T_7 = \{(b_p, b, b_i), (b_p, a, b_p), (b_p, c, b_p), (b_i, b, b_p), (b_i, a, b_i), (b_i, c, b_i)\}$

— $L_{81} = L_6 \cup L_7$ (nommez les états de façon pertinente.

Correction

$\mathcal{A}_{81} = (\Sigma, Q_{81}, a_pb_p, F_{81}, T_{82})$ avec $Q_{81} = \{a_pb_p, a_pb_i, a_ib_p, a_ib_i\}$, $F_{81} = \{a_pb_p, a_pb_i, a_ib_i\}$ et

$$T_{81} = \{(a_pb_p, a, a_ib_p), (a_pb_i, a, a_ib_i), (a_ib_p, a, a_pb_p), (a_ib_i, a, a_pb_i), (a_pb_p, b, a_pb_i), (a_ib_p, b, a_ib_i),$$

$$(a_pb_i, b, a_pb_p), (a_ib_i, b, a_ib_p)\} \cup \{(x, c, x) \mid x \in Q_8\}$$

— $L_{82} = L_6 \cap L_7$. Nommez les états de façon pertinente.

Correction

Pour obtenir L_{82}, il nous suffit de reprendre A_{81} en mettant comme ensemble d'état finaux $F_{82} = \{a_pb_i\}$.

— L_9, l'ensemble des mots ayant un nombre de a multiple de 41. Donner une description formelle ;

Correction

$\mathcal{A}_9 = (\Sigma, Q_9, q_0, F_9, T_9)$ avec $Q_9 = \{q_0, q_1, ..., q_{40}\}$, $F_9 = \{q_0\}$ et

$$T_9 = \{\forall i \in 0..39, (q_i, a, q_{i+1})\} \cup \{(q_{40}, a, q_0)\} \cup \{\forall i \in 0..40, (q_i, b, q_i)\}$$

— $L_{10} = L_6 \cup L_9$.

Exercice 1.8 *Automates plus complexes*

— Donner un automate qui reconnait le code `2341` sur un digicode à quatre touches : $\{1, 2, 3, 4\}$;

— même question pour le code `2232` ;

— construire un automate reconnaissant les nombres multiples de 3 écrits en base 10 ;

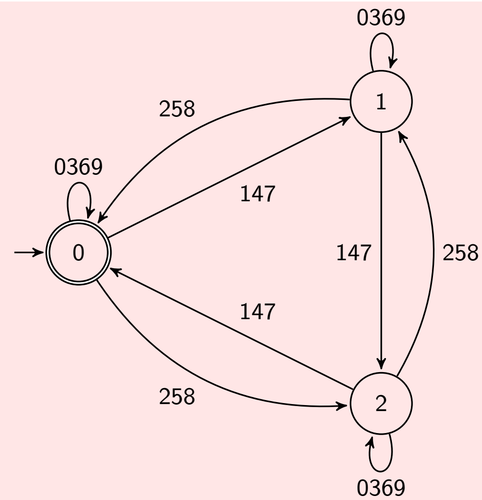
Correction

Le critère de divisibilité par 3 des nombres écrits en base 10 est que la somme des chiffres est divisible par 3. C'est lié au fait que toutes les puissance de 10 sont des multiples de 3 plus 1.

Les nombres sont vus comme des mots sur l'alphabet $\Sigma = 0, 1, ..., 9$et sont lus par l'automate de gauche à droite.

Pour les états on ne va pas stocker la somme des chiffres (ce qui nécessiterait une infinité d'états) mais uniquement la valeur de cette somme modulo 3 qui peut prendre trois valeurs : 0,1,2.

L'état 0 est à la fois état initial et final.



Exemple : La lecture du "mot" 5413, passera par les états 0, 2, 0, 1, 1. La lecture se termine dans l'état 1 qui n'est pas final. Donc 5413 n'est pas multiple de 3. Le fait de terminer dans l'état 1 indique que 5413 modulo 3 vaut 1.

— construire un automate reconnaissant les nombres multiples de 3 écrits en base 2 ;

Correction

Les puissances paires de 2 sont égales à 1 modulo 3 : $2^0 = 1$, $2^2 = 3+1$, $2^4 = 3 \times 5+1$. Les puissances impaires de 2 sont égales à 2 modulo 3 : $2^1 = 2$, $2^3 = 6 + 2$, $2^5 = 30 + 2$. Que l'on peut représenter par -1 modulo 3 : $2^1 = 3 - 1$, $2^3 = 9 - 1$, $2^5 = 33 - 1$.

Dans la représentation binaire, chaque 1 sur une puissance paire vaudra $+1$ et chaque 1 sur une puissance impaire vaudra -1. En faisant la somme de ces $+1$ et -1 et en prenant cette somme modulo 3, on aura $+1$, 0 et -1.

Lire un 0 de plus change tous les $+1$ en -1 et vice-versa. Si la somme valait 0, elle reste à 0. Si la somme valait $+1$ elle devient -1 et vice versa.

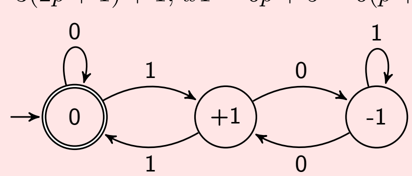
Lire un 1 de plus change tous les $+1$ en -1 et vice-versa sur les chiffres déjà lus et ajoute un $+1$ qui est le 2^0.

Une autre manière de voir est de dire que si on a un nombre x en binaire et qu'on lui rajoute un 0 à droite (noté $x0$) on le multiplie par 2. Si on rajoute un 1 à droite (noté $x1$), on le multiple par 2 et on ajoute 1.

— si $x = 3p$, $x0 = 6p$, $x1 = 6p + 1$

— si $x = 3p + 1$, $x0 = 6p + 2 = 6(p + 1) - 1$, $x1 = 6p + 2 + 1 = 6(p + 1)$

— si $x = 3p + 2$, $x0 = 6p + 4 = 3(2p + 1) + 1$, $x1 = 6p + 5 = 6(p + 1) - 1$



Exemple : La lecture du "mot" 1001101, se termine dans l'état -1, ce qui signifie que 1001101 est congru à -1 modulo 3.

— construire un automate reconnaissant les entiers signés en langage C.

Exercice 1.9 *Le Loup, la Chèvre et le Chou*
Construire un automate permettant de résoudre le problème du Loup, de la Chèvre et du Chou, voir : `http://fr.wikipedia.org/wiki/Problemes_de_passage_de_riviere` ;

Correction

Le problème est modélisé par un automate.

Les états

Chaque état de l'automate représente une photographie du système. La barre verticale représente la rivière. Un • représente la position du fermier et de la barque (qui sont toujours ensemble). L représente la position du Loup (à gauche ou à droite de la rivière), C est la position de la chèvre (à gauche ou à droite), S représente la Salade qui remplace le Chou (à gauche ou à droite).

Il y a 16 état possibles, car chacun des 4 "personnages" (•, L, C, S) a deux positions possibles (à gauche ou à droite de la rivière) donc en tout $2^4 = 16$ états.

Il y a des états interdits quand L et C (ou C et S)ou sont du même côté et sans le fermier. Ces états sont colorés en rouge ci-dessous.

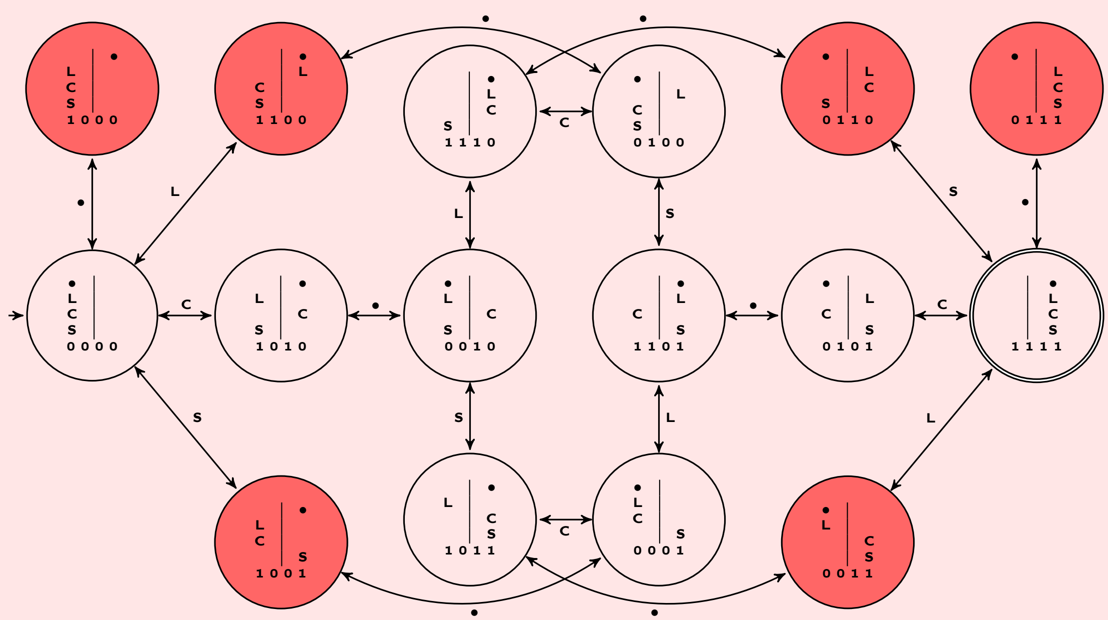
L'état de départ est l'état avec tout à gauche et l'état d'arrivée est l'état avec tout à droite.

Les transitions

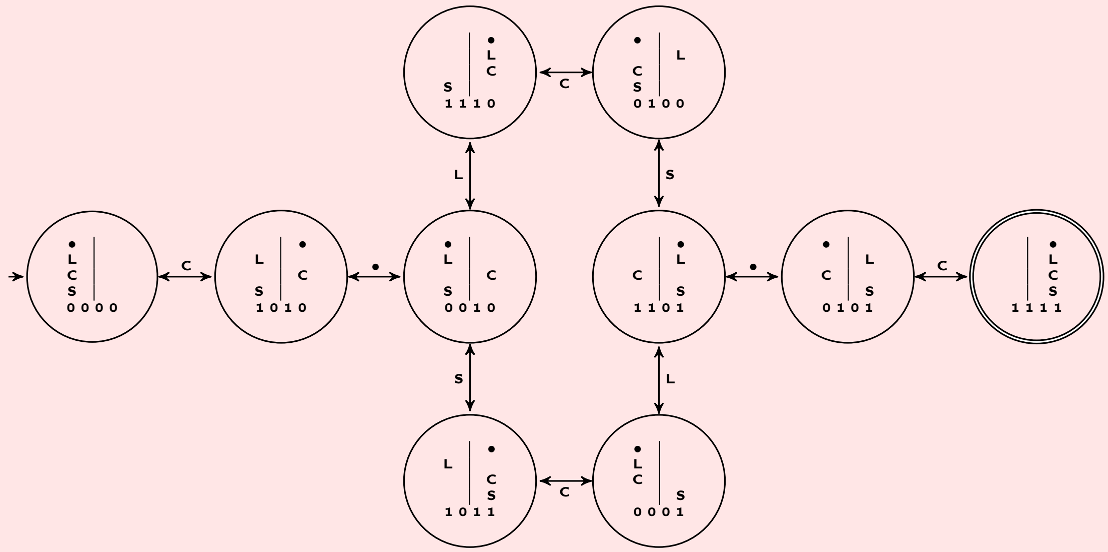
Une transition entre deux états représente une traversée de la rivière. Donc • change de côté à chaque transition. Si le fermier est seul dans la barque, la transition a comme label •. Quand le fermier traverse avec un des trois éléments L, C ou S, la transition a comme label l'élément qui traverse.

S'il y a une transition d'un état vers un autre, la transition dans l'autre sens existe aussi avec le même label. Dans les dessins ci-dessous, on représente ces deux transitions par un seul trait avec une flèche à chaque bout. Formellement c'est deux transitions.

Automate complet



Après suppression des états interdits



Séquence d'états résolvant le problème

Une fois cette modélisation par automate effectuée, les solutions du problème initial sont les mots reconnus par l'automate. La solution la plus efficace consistera à trouver le ou les mots les plus courts reconnus par l'automate. Il suffit donc de trouver les chemins les plus courts allant de l'état initial à l'état final.

Il y a deux mots de longueur minimale (7 transitions) qui vont de l'état initial à l'état final :

— C • L C S • C

— C • S C L • C

Si les "personnages" ne se mangeaient pas entre eux, faire traverser tout le monde se ferait en 5 transitions, ce que l'on voit sur l'automate complet en passant par des états rouges.

Pour aller plus loin

On peut noter que le Loup et la Salade jouent un rôle symétrique et que transformer le problème avec toujours une seule chèvre et en mettant deux salades et pas de Loup (ou bien deux Loups et pas de salade) diminue le nombre d'états.

Exercice 1.10 *Les bidons de trois et cinq gallons*

Construire un automate permettant de résoudre le problème des bidons de trois gallons et de cinq gallons, voir : https://www.youtube.com/watch?v=pmk2mNf9iqE

Correction

Le but est de mettre exactement 4 gallons dans le bidon de cinq. On peut remplir, vider ou transvaser les bidons.

Les états

Le problème est modélisé par un automate. Un état représentera le contenu des deux bidons (celui de trois galons, puis celui de cinq galons). Par exemple l'état 2|4 signifie qu'il y a 2 galons dans le bidon de trois et 4 galons dans le bidon de cinq.

Le bidon de trois peut prendre quatre valeurs différents (0, 1, 2, 3) et le bidon de cinq peut prendre six valeurs différents (0 à 5). Il y a donc 24 états possible.

L'état initial

L'état initial est 0|0, les deux bidons sont vides.

Les états finaux

Les quatre états x|4 sont des états finaux.

Les transitions

Il y a six transitions possibles :

— remplir le bidon de trois notée +3 ;

— remplir le bidon de cinq notée +5 ;

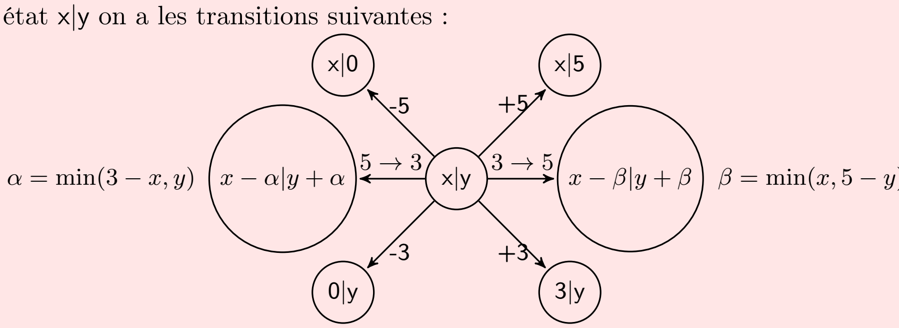
— vider par terre le bidon de trois notée -3 ;

— vider par terre le bidon de cinq notée -5 ;

— transvaser tout ce que l'on peut du bidon de trois vers le bidon de cinq notée 3→5 ;

— transvaser tout ce que l'on peut du bidon de cinq vers le bidon de trois notée 5→3 ;

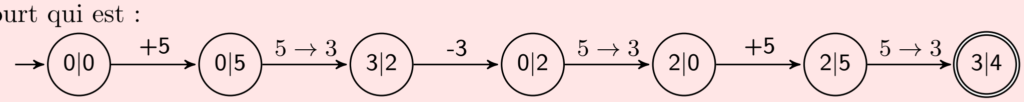
Ce qui donne comme alphabet : $\Sigma = \{+3, -3, +5, -5, 3 \to 5, 5 \to 3\}$.
Pour tout état x|y on a les transitions suivantes :



Certaines transitions sont en fait des boucles, par exemple pour tous les états x|5, la transition +5 fait une boucle.

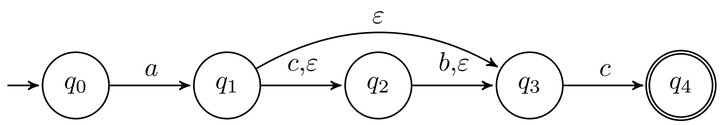
La solution

La solution consiste à trouver un mot reconnu par l'automate. La solution la plus efficace sera le mot le plus court qui est :



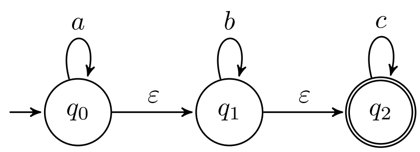
2 TD 2 – Manipulation d'automate - 1 (avec corrigés)

Exercice 2.1 *Suppression des ε-transitions*
Soit l'automate :



1. le langage reconnu par cet automate est-il fini ou infini ?
2. supprimer les ε-transitions de l'automate ;
3. déterminiser l'automate.
4. quel est le langage reconnu par l'automate ?
5. à partir des mots du langage, construire un automate en mettant un branche par mot.
6. déterminiser ce nouvel automate. Est-il le même que celui trouvé au point 3 ?

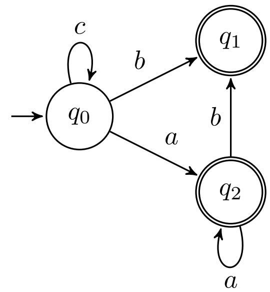
Exercice 2.2 *Suppression des ε-transitions*
Soit l'automate :



1. le langage reconnu par cet automate est-il fini ou infini ?
2. supprimer les ε-transitions de l'automate ;
3. déterminiser l'automate ;
4. quel est le langage reconnu par cet automate ?

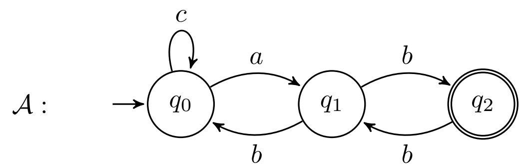
Exercice 2.3 *AFD*
Soit l'automate :

$\mathcal{A}$:



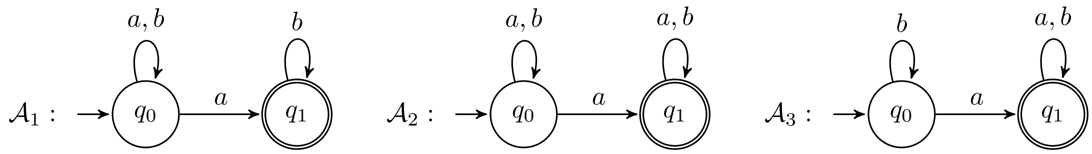
1. donnez sa description formelle $\mathcal{A} = (\Sigma, Q, q_0, F, T)$;
2. les mots $aaab$, cac, cc sont-ils reconnus par $\mathcal{A}$?
3. est-il déterministe, si non déterminisez-le ?
4. donner la description formelle de l'automate déterminisé ;
5. est-il complet, si non rendez-le complet ?
6. donner la description formelle de l'automate complet.

Exercice 2.4 *AFD*
Soit l'automate :



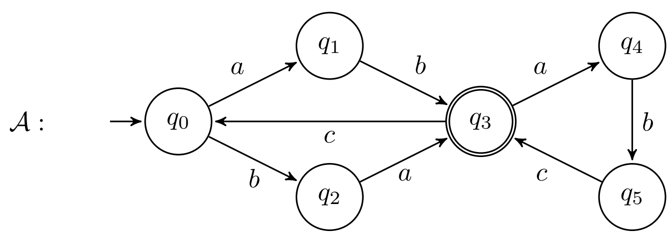
1. donnez sa description formelle $\mathcal{A} = (\Sigma, Q, q_0, F, T)$;
2. les mots *abbb*, *cabb* sont-ils reconnus par $\mathcal{A}$?
3. est-il déterministe, si non déterminisez-le ?
4. est-il complet, si non rendez-le complet ?

Exercice 2.5 *AFD*
Soit l'automate :



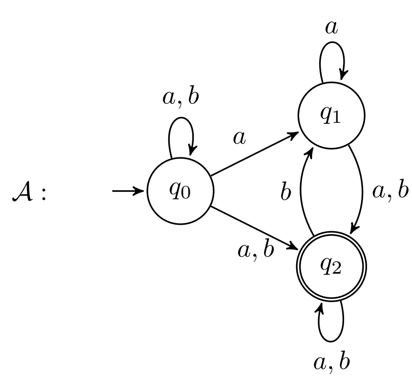
1. quelles différences entre les trois automates ?
2. quels automates sont déterministes ?
3. déterminisez ceux qui ne le sont pas ?
4. quel est le langage reconnu par chacun des automates

Exercice 2.6 *AFD*
Soit l'automate :



1. donnez sa description formelle $\mathcal{A} = (\Sigma, Q, q_0, F, T)$;
2. est-il déterministe, si non déterminisez-le ?
3. est-il complet, si non rendez-le complet ?

Exercice 2.7 *AFD*
Soit l'automate :

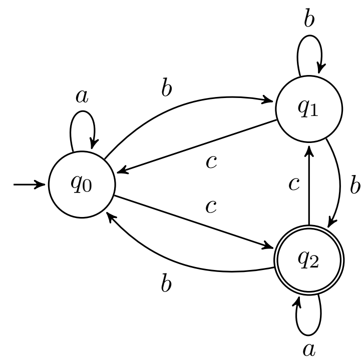


1. est-il déterministe, si non déterminisez-le ?

2. est-il complet, si non rendez-le complet ?
3. quel est le langage reconnu par l'automate ?

Exercice 2.8 *AFD*
Soit l'automate :

$\mathcal{A}$:



1. donnez sa description formelle $\mathcal{A} = (\Sigma, Q, q_0, F, T)$;
2. est-il déterministe, si non déterminisez-le ?
3. est-il complet, si non rendez-le complet ?

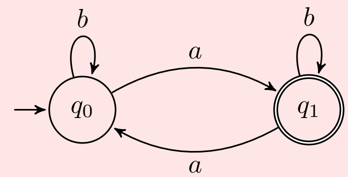
3 TD 3 – Manipulation d'automate - 2 (avec corrigés)

Dans cette section, on définit sur l'alphabet $\Sigma = \{a, b\}$, les langages suivants :

— L_1 l'ensemble des mots ayant un nombre impair de a ;

Correction

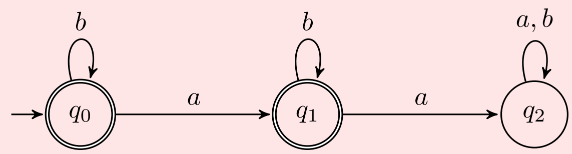
$\mathcal{A}_1$:



— L_2 l'ensemble des mots ayant au plus un a ;

Correction

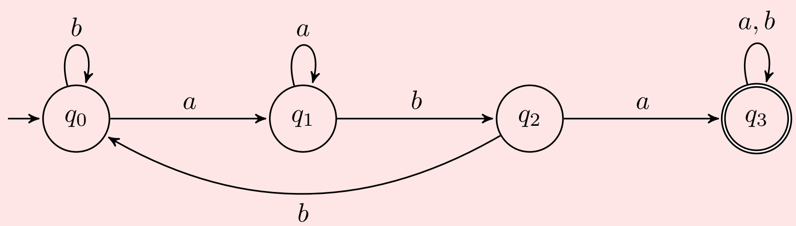
$\mathcal{A}_2$:



— L_3 l'ensemble des mots ayant aba comme sous-mot.

Correction

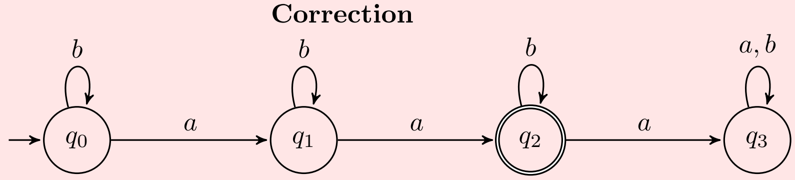
$\mathcal{A}_3$:



— L_4 l'ensemble des mots ayant exactement deux a.

Correction

$\mathcal{A}_4$:



Exercice 3.1 *Construction d'automate*
Donner un automate déterministe complet reconnaissant chacun des langages L_1 à L_4.

Correction

Voir ci-dessus

Exercice 3.2 *Langage complémentaire*
On note $\overline{L_1} = \Sigma^* \backslash L_1$ et $\overline{L_2} = \Sigma^* \backslash L_2$

— Exprimer en français le contenu de ces langages.

— Construire les automates reconnaissant $\overline{L_1}$ et $\overline{L_2}$ à partir de ceux reconnaissant L_1 et L_2.

— Pour tout automate fini déterministe complet, donner une définition formelle de l'automate reconnaissant le langage complémentaire.

Correction

Si le langage L_x est reconnu par l'automate **déterministe complet** $\mathcal{A}_x = (\Sigma_x, Q_x, q_{x0}, F_x, T_x)$ alors le langage $\overline{L_x} = \Sigma^* \backslash L_x$ est reconnu par l'automate $\mathcal{A} = (\Sigma, Q, q_0, F, T)$ avec :

- $\Sigma = \Sigma_x$, l'alphabet est le même ;
- $Q = Q_x$, l'ensemble des états est le même ;
- $q_0 = q_{x0}$, l'état initial est le même ;
- $F = Q_x \backslash F_x$, les états finaux sont les états qui n'étaient pas finaux dans $\mathcal{A}_x$ et vice-versa ;
- $T = T_x$, l'ensemble des transition est le même.

Attention, cet algo ne marche que si l'automate de L est **déterministe** et **complet**. Le résultat est un automate déterministe complet.

Exercice 3.3 *Union de langages*

On note $L_{13} = L_1 \cup L_3$.

- Construire l'automate reconnaissant L_{13} à partir des automates reconnaissant L_1 et L_3.
- Pour tout couple d'automates, donner une définition formelle de l'automate reconnaissant l'union des deux langages.

Correction

Si le langage L_x est reconnu par l'automate $\mathcal{A}_x = (\Sigma_x, Q_x, q_{x0}, F_x, T_x)$
et le langage L_y est reconnu par l'automate $\mathcal{A}_y = (\Sigma_y, Q_y, q_{y0}, F_y, T_y)$
alors le langage $L = L_x \cup L_y$ est reconnu par l'automate $\mathcal{A} = (\Sigma, Q, q_0, F, T)$ avec :

- $\Sigma = \Sigma_x \cup \Sigma_y$, souvent les deux alphabets Σ_x et Σ_y sont les mêmes, mais ce n'est pas formellement obligatoire ;
- $Q = Q_x \cup Q_y \cup \{q_0\}$, en plus des états des deux automates, on ajoute un nouvel état qui est l'état initial de l'automate de l'union ;
- q_0, l'état initial est un état supplémentaire qui n'est ni dans Q_x ni dans Q_y ;
- $F = F_x \cup F_y$, les états finaux sont les états finaux des deux automates ;
- $T = T_x \cup T_y \cup \{(q_0, \varepsilon, q_{x0}), (q_0, \varepsilon, q_{y0})\}$, on ajoute des ε-transitions depuis le nouvel état initial vers les anciens états initiaux.

Les automates de départ peuvent être déterministes ou non. Avec l'ajout des ε-transitions, l'automate résultant n'est pas déterministe. On peut alors faire passer les algos de suppression d'ε-transition et de déterminisation.

Il est à noter que l'on peut faire l'union simultanée d'autant de langages que l'ont veut en ajoutant à chaque fois une ε-transition de q_0 vers les états iniiaux des automates dont on fait l'union.

Exercice 3.4 *Intersection de langages*

- Donner une relation ensembliste permettant de définir l'intersection de deux ensembles à partir de l'union et du complémentaire.

Correction

Loi de DE MORGAN : Pour tous ensembles A et B, on a :

- $\overline{A \cup B} = \overline{A} \cap \overline{B}$
- $\overline{A \cap B} = \overline{A} \cup \overline{B}$, en prenant le complémentaire des deux côtés, on a : $A \cap B = \overline{\overline{A} \cup \overline{B}}$

- En déduire un algo de construction de l'automate reconnaissant l'intersection de deux langages.

Correction

Si on a deux automates $\mathcal{A}_x$ et $\mathcal{A}_y$, on calcule d'abord les automates qui reconnaissance le complémentaire de chacun des deux automates, ensuite on fait l'union, puis on calcule le complémentaire de cette union.

Exercice 3.5 *Intersection de langages par produit*
On note $L_{12} = L_1 \cap L_2$.

— Donner un algo pour construire l'automate de l'intersection des langages L_1 et L_2 en utilisant un automate produit.

— Construire l'automate reconnaissant L_{12} à partir des automates reconnaissant L_1 et L_2.

— Pour tout couple d'automates, donner une définition formelle de l'automate reconnaissant l'intersection des deux langages.

Exercice 3.6 *Concaténation de deux langages*
On note $L_{34} = L_3 \cdot L_4$.

— Construire l'automate reconnaissant L_{34} à partir des automates reconnaissant L_3 et L_4.

— Pour tout couple d'automates, donner une définition formelle de l'automate reconnaissant la concaténation des deux langages.

Correction

Si le langage L_x est reconnu par l'automate $\mathcal{A}_x = (\Sigma_x, Q_x, q_{x0}, F_x, T_x)$
et le langage L_y est reconnu par l'automate $\mathcal{A}_y = (\Sigma_y, Q_y, q_{y0}, F_y, T_y)$
alors le langage $L = L_x \cdot L_y$ est reconnu par l'automate $\mathcal{A} = (\Sigma, Q, q_0, F, T)$ avec :

— $\Sigma = \Sigma_x \cup \Sigma_y$, souvent les deux alphabets Σ_x et Σ_y sont les mêmes, mais ce n'est pas formellement obligatoire ;

— $Q = Q_x \cup Q_y$;

— $q_0 = q_{x0}$, l'état initial est l'état initial du premier automate ;

— $F = F_y$, les états finaux sont les états finaux du deuxième automate ;

— $T = T_x \cup T_y \cup \{(q, \varepsilon, q_{y0}), \forall q \in F_x\}$, on ajoute des ε-transitions depuis les états finaux du premier automate vers l'état intial du deuxième.

Les automates de départ peuvent être déterministes ou non. Avec l'ajout des ε-transitions, l'automate résultant n'est pas déterministe. On peut alors faire passer les algos de suppression d'ε-transition et de déterminisation.

Exercice 3.7 *Étoile d'un langage*

Correction

On rappelle que l'étoile de Kleene d'un langage notée L^* est définie par :

$$L^* = \bigcup_{n=0}^{+\infty} L^n$$

avec L^n qui est la concaténation n fois du langage L avec lui même, par exemple $L^2 = L \cdot L$, $L^1 = L$ et $L^0 = \{\varepsilon\}$.

— Construire l'automate reconnaissant L_3^* à partir de l'automate reconnaissant L_3.

— Construire l'automate reconnaissant L_4^* à partir de l'automate reconnaissant L_4.

— Pour tout automate, donner une définition formelle de l'automate reconnaissant l'étoile du langage.

Correction

On va noter :

$$L^+ = \bigcup_{n=1}^{+\infty} L^n$$

avec cette notation, on a $L^* = L^+ \cup \{\varepsilon\}$ (l'union commence à $n = 1$ et pas à $n = 0$ comme pour L^*).
Si le langage L est reconnu par l'automate $\mathcal{A} = (\Sigma, Q, q_0, F, T)$
alors le langage L^+ est reconnu par l'automate $\mathcal{A}' = (\Sigma', Q', q_0', F', T')$ avec :

— $\Sigma' = \Sigma$, l'alphabet est le même ;

— $Q' = Q$, l'ensemble des états est le même ;

— $q_0' = q_0$, l'état initial est le même ;

— $F' = F$, les états finaux sont les mêmes ;

— $T' = T \cup \{(q, \varepsilon, q_0), \forall q \in F\}$, on rajoute des ε-transitions depuis les états finaux vers l'état initial.

L'automate de départ peut-être non déterministe. Le résultat est un automate non déterministe à cause des ε-transitions. Comme d'hab, on peut supprimer les ε-transitions et le déterminiser.
Pour l'automate qui reconnaît L^*, il faut faire l'union du langage L^+ et du langage $\{\varepsilon\}$. Si q_0 est un état final, il n'y a rien à faire, sinon on fait l'union comme vu à l'exercice 3.3 ou plus simplement on ajoute un nouvel état initial qui est aussi final et qui a une transition avec ε vers l'état q_0.

Exercice 3.8 *Rationnel ⇒ Reconnaissable*

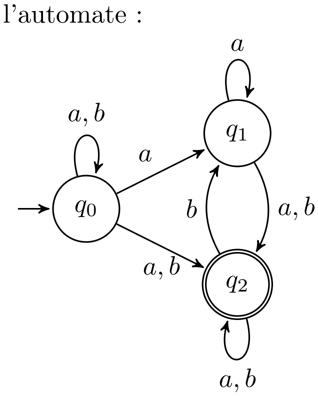
— donner un automate qui reconnait $L = \emptyset$;
— donner un automate qui reconnait $L = \{\epsilon\}$;
— donner un automate qui reconnait $L = \{a\}$;
— rappeler la définition d'un langage rationnel ;
— avec les exercices 3.2, 3.3, 3.7, 3.6 et les résultats ci-dessus, montrer que si un langage est rationnel, il est reconnaissable.

Exercice 3.9 *Algorithme de Brzozowski de minimisation d'un automate fini*
Soit $\mathcal{A} = (\Sigma, Q, q_0, F, T)$ un automate. L'automate transposé de $\mathcal{A}$ noté $\mathrm{tr}(\mathcal{A}) = (\Sigma', Q', q_0', F', T')$ est défini par :
— $\Sigma' = \Sigma$;
— $Q' = Q \cup \{q_0'\}$;
— $F' = \{q_0\}$;
— $T' = \{(q, a, p) \text{ telle que } (p, a, q) \in T\} \bigcup \{(q_0', \varepsilon, q) \text{ avec } q \in F\}$.
Pour tout automate $\mathcal{A}$, on note $\det(\mathcal{A})$ l'automate déterministe reconnaissant le même langage que $\mathcal{A}$.

L'algorithme de Brzozowski consiste à calculer $\det(\mathrm{tr}(\det(\mathrm{tr}(\mathcal{A}))))$, qui est alors l'automate minimum reconnaissant le même langage que $\mathcal{A}$.

Appliquer l'algorithme de Brzozowski sur l'automate :

$\mathcal{A}$:



Minimiser l'automate avec l'algo vu en cours.

Exercice 3.10 *Minimisation*
Soit L_0, le langage des mots ayant un nombre pair de a.

1. Donner un automate reconnaissant L_0.
2. Donner un automate reconnaissant $L_0 \cup L_1$, en le construisant comme vu dans l'exercice 3.3.
3. Supprimer les ε-transitions de l'automate du point 2.
4. Déterminiser l'automate du point 3.
5. Minimiser l'automate du point 4.

Même exercice avec l'automate de $L_0 \cdot L_1$.

4 TD 4 – Expression régulière

Exercice 4.1 *Expression régulière*
Soit l'alphabet $\Sigma = \{a, b, c\}$, donnez les expressions régulières pour les langages suivants :

- les mots ne commençant pas par un c ;
- les mots contenant deux a consécutifs ;
- les mots contenant à la fois les facteurs (sous-mots) aa et bb ;
- les mots dont la première et la dernière lettre sont les mêmes ;
- les mots ne contenant pas deux a consécutifs.

Exercice 4.2 *Expression régulière*
Donnez les expressions régulières pour les langages suivants, préciser l'alphabet à chaque fois :

- les entiers écrits en binaire ;
- les entiers impairs en binaire ;
- les entiers divisibles par 5 en base 10 ;
- les entiers divisibles par 4 en base 10 ;
- les noms des variables en `C`.

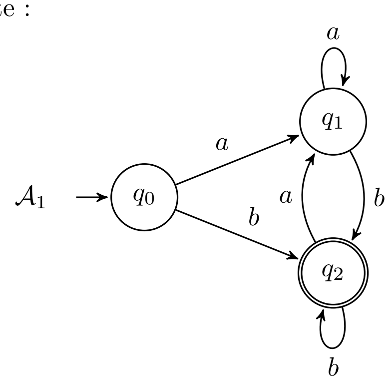
Exercice 4.3 *Fini et régulier*
Montrer que tous les langages finis sont réguliers.

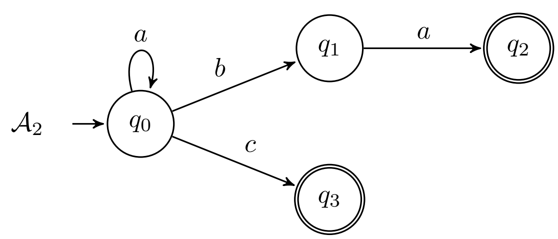
Exercice 4.4 *e.r. vers automate*
Donnez un automate reconnaissant les langages définis par les e.r. suivantes :

- $e_1 = (a + b)^*$;
- $e_2 = \varepsilon$;
- $e_3 = \emptyset$;
- $e_4 = a^*b$;
- $e_5 = ba^*$;
- $e_6 = a^*b + ba^*$;
- $e_7 = (a^*b + ba^*)^*$.

Exercice 4.5 *Égalité d'e.r.*
Montrer que les deux expressions régulières suivantes sont équivalentes : $e_1 = (a + b)^*$ et $e_2 = (ab^*)^* + (ba^*)^*$.

Exercice 4.6 *AFN vers e.r.*
Pour chaque automate, calculer les expressions régulières dont le langage est le même que celui reconnu par l'automate :





Exercice 4.7 *La totale*
Soit le langage qui est l'ensemble des mots contenant une seule occurence de aaa sur l'alphabet $\Sigma = \{a, b\}$:

1. donner une expression régulière de ce langage ;
2. construire l'AFN avec ε-transition reconnaissant ce langage ;
3. déterminiser, compléter et minimiser l'automate ;
4. calculer l'expression régulière de l'automate minimum ;
5. comparer avec l'expression régulière calculée au 1.

Exercice 4.8 *True Lies*
Pour les langages suivants, donner une expression régulière, l'AFN, l'AFD et calculer l'e.r. à partir de l'automate :

1. les mots de longueur paire sur l'alphabet $\{a, b\}$;
2. les mots ne contenant pas ba sur l'alphabet $\{a, b\}$;
3. les mots dont la dernière lettre est au moins deux fois dans le mot sur l'alphabet $\{a, b, c\}$;
4. les mots dont la dernière lettre n'est pas ailleurs dans le mot dans le mot sur l'alphabet $\{a, b, c\}$.

5 TD 5 – Lemme de l'étoile

Utilisation de lemme de l'étoile pour prouver qu'un langage n'est pas régulier :

Rappel du lemme

SI un langage L est régulier **ALORS** $\exists N > 0$, tel que $\forall z \in L, |z| > N$:

— $\exists u, v, w$ tels que $z = uvw$;

— $|v| > 0$;

— $|uv| \leq N$;

— $\forall i \geq 0$, $uv^i w \in L$.

Exemple

On va montrer que $L = a^n b^n$ n'est pas régulier par contradiction :

1. **Choix du mot** z **:** On suppose que L est régulier et on considère le mot $z = a^N b^N$, on a bien $|z| > N$ et la contrainte $|uv| \leq N$ fait que la décomposition z en uvw est telle que $u = a^x$, $v = a^y$ et $w = a^z b^N$ avec $y > 0$, $x + y < N$ et $x + y + z = N$.
2. **Choix du** i **:** D'après le lemme de l'étoile uv^2w doit appartenir au langage, or $uv^2w = a^{N+y}b^N$ avec $y > 0$ et donc $uv^2w \notin L$, ce qui est une contradiction.
3. **Conclusion :** Donc l'hypothèse de départ que L est régulier est fausse et donc L n'est pas régulier.

La technique de preuve consiste donc à trouver le mot z et la valeur de i astucieux pour obtenir une contradiction pour toutes les décompositions de z.

Exercice 5.1 *Langage non régulier*
Montrez que les langages suivants ne sont pas réguliers :

— $L_1 = \{a^n b^p, 0 \leq n < p\}$ (indication : $z = a^N b^{N+k}$) ;

— $L_2 = \{a^{n^2}, n \geq 0\}$;

— $L_3 = \{a^n b^p, 0 \leq n \neq p\}$ (indication : par l'absurde en utilisant le complémentaire de $a^n b^n$).

Exercice 5.2 *Régulier ou pas*
Pour les langages suivants prouver s'ils sont réguliers ou pas :

— $L_4 = \{a^p, p \text{ premier}\}$;

— $L_5 = \{a^n b^p, n \equiv p \pmod 2\}$;

— $L_6 = \{a^n b^p, n \geq p\}$;

— $L_7 = \{w, |w|_a = |w|_b\}$;

6 TD 6 – Automate à pile

Exercice 6.1 *Automate à pile*
Donner les automates à pile reconnaissant les langages suivants, en précisant si la reconnaissance s'effectue par pile vide ou par état final. Les automates ne doivent pas être nécessairement déterministes

— $L_1 = \{a^n b^n, n \geq 0\}$;
— $L_2 = \{a^i b^j c^k, i, j, k \geq 0 \text{ et } (i = j \text{ ou } j = k)\}$.

Exercice 6.2 *Automate à deux piles*
Pour $\Sigma = \{a, b\}$, donner les automates à une ou deux piles reconnaissant les langages suivants :

— L_3 : l'ensemble des palindromes ;
— $L_4 = \{w \cdot w, w \in \Sigma^*\}$;
— $L_5 = \{a^{n^2}, n \geq 0\}$.

Exercice 6.3 *Grammaire*
Construire les grammaires pour les langages suivants :

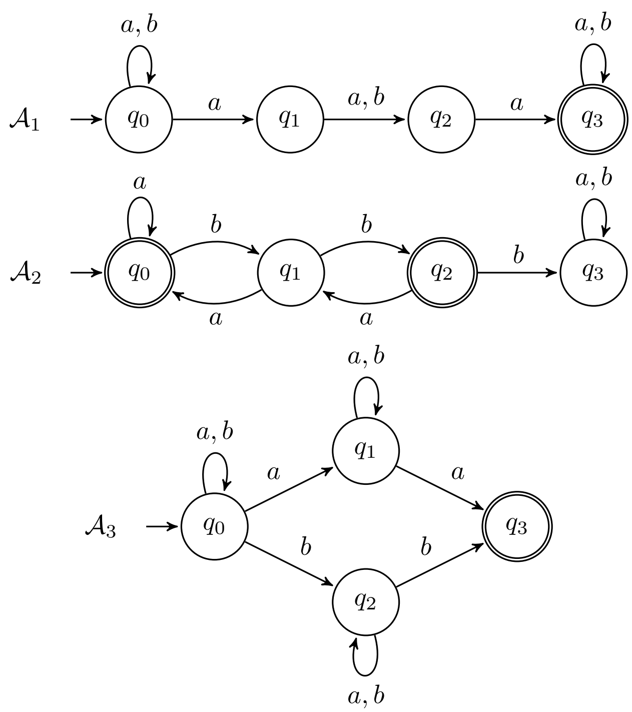
— L_6 : a^*b ;
— L_7 : ab^* ;
— L_8 : $(a^*b + ab^*)aa$;
— L_9 : $((a^*b + ab^*)aa)^*$;
— $L_{10} = \{a^n bc^n, n > 0\}$;
— $L_{11} = \{a^{n+1} b^n, n > 0\}$;
— $L_{12} = \{a^i b^j c^j b^i, i, j > 0\}$.

7 TD 7 – Grammaire

Exercice 7.1 *Grammaire*
Construire les grammaires pour les langages suivants en donnant à chaque fois la définition formelle :

- L_1 : a^*b ;
- L_2 : ab^* ;
- L_3 : $(a^*b + ab^*)aa$;
- L_4 : $((a^*b + ab^*)aa)^*$;
- $L_5 = \{a^nbc^n, n > 0\}$;
- $L_6 = \{a^{n+1}b^n, n > 0\}$;
- $L_7 = \{a^ib^jc^jb^i, i, j > 0\}$.
- $L_8 = \{a^nb^nc^n, n \geq 0\}$.

Exercice 7.2 *Automate vers grammaire*
Donner la grammaire engendrant le même langage que les automates suivants (en donnant à chaque fois la définition formelle) :



Exercice 7.3 *Ambiguïté*

1. Montrer qu'une grammaire **régulière droite** est **contextuelle** de type 1.
2. Montrer qu'une grammaire **non ambiguë** est $LL(1)$.
3. Montrer qu'une grammaire est $LL(1)$ si elle n'est **pas ambiguë**.
4. Pour chacune des grammaires des deux exercices précédents, préciser sont type, si elle est **régulière droite**, **ambiguë**

Exercice 7.4 *Formes normales*
Les deux liens suivants sont à consulter et à étudier pour cet exercice :

— https ://fr.wikipedia.org/wiki/Forme_normale_de_Greibach

— https ://fr.wikipedia.org/wiki/Forme_normale_de_Chomsky

Soit la grammaire $G = (\Sigma, V, S, \mathcal{P})$ avec :

— $\Sigma = \{a, b\}$

— $V = \{S\}$

— $\mathcal{P} = \{$

$S \rightarrow aSbS,$

$S \rightarrow a,$

$S \rightarrow b,$

$S \rightarrow \varepsilon$

$\}$

1. Mettre G sous forme normale de Greibach, ce qui donne la grammaire G_1. L'algorithme est donné dans la section Construction de la page : https ://fr.wikipedia.org/wiki/Forme_normale_de_Greibach
2. Mettre G sous forme normale de Chomsky, ce qui donne la grammaire G_2 ; L'algorithme est donné dans la section Conversion de la page : https ://fr.wikipedia.org/wiki/Forme_normale_de_Chomsky
3. Mettre la grammaire G_1 sous forme normale de Chomsky, ce qui donne la grammaire G_3 ;
4. Comparer G_2 et G_3, que constatez-vous ?

8 TD 8 – Machine de Turing - 1

Exercice 8.1 *Premières machines*
Ecrire des machines de Turing suivantes sur l'alphabet $\Sigma = \{0, 1\}$:

1. définir les nombres en base un (unaire) ;
2. donner une machine de Turing réalisant l'addition en base un ;
3. reconnaître si un mot (nombre) en binaire est pair ;
4. renverser un mot ;
5. ajouter 1 à un nombre en binaire.

Plusieurs rubans
On peut définir une machine de Turing à k rubans de façon similaire à une machine de Turing à un ruban. Si Q est l'ensemble d'états de la machine et Γ l'alphabet des rubans, la fonction de transitions est alors de la forme

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\mathrm{G}, \mathrm{D}\})^k .$$

On remarquera que chacune des têtes de lecture bouge indépendemment et que la transition dépend conjointement des lettres lues sur l'ensemble des rubans. La configuration initiale d'une telle machine est l'entrée écrite sur le premier ruban et tous les autres rubans vides.

Exercice 8.2 *Machines plus dures*
Ecrire des machines de Turing reconnaissant les langages suivants :

1. $\{a^n b^n c^n, n \in N^*\}$ écrire la machine avec un, puis deux rubans ;
2. $\{wcw^R, w \in \{a, b\}^*\}$ (w^R est le mot miroir de w) utiliser un seul ruban ;
3. $\{a^n b^{n^2}, n \in N\}$ utiliser trois rubans.

Exercice 8.3 *Machines de Turing qui calculent*

1. $a^n \to a^n b^n, n \geq 0$;
2. $w \to ww^R$ (w^R est le mot miroir de w) ;
3. $w \to ww$;
4. addition en binaire ;
5. soustraction en binaire ;
6. $n \to n^2$.

Exercice 8.4 *Quelques exemples*
Donner les machines de Turing reconnaissant les langages suivants

- $L_1 = \{a^i b^j a^i b^j, i, j \in \mathbb{N})\}$;
- $L_2 = \{w.w \mid w \in \{a, b\}^*\}$ (attention, pas de délimiteur entre les deux copies du mot) ;
- $L_3 = \{1^n \# 1^{n^2} | n > 0\}$.

Exercice 8.5 *Plusieurs rubans*
Donner des machines de Turing reconnaissant les langages suivants

- $L_4 = \{i\#j\#k | i, j, k$ entiers codés en binaire avec $k = i + j\}$ avec trois rubans ;
- $L_5 = \left\{a^{n^2} \mid n \in \mathbb{N}\right\}$ avec trois rubans ;
- $L_6 = \{i\#j\#k | i, j, k$ entiers codés en binaire avec $k = i \times j\}$ avec quatre rubans.

Exercice 8.6 *Automates et machines de Turing*
Montrer que tout langage régulier est décidable par une machine de Turing. Montrer que tout langage reconnaissable par un automate à pile déterministe est reconnaissable par une machine de Turing.

9 TD 9 – Machine de Turing - 2

Exercice 9.1 *Ruban bi-infini*
Une machine de Turing à ruban bi-infini est une machine de Turing dont le ruban est infini vers la gauche et vers la droite – par opposition à la définition standard où le ruban est infini seulement vers la droite. Montrer que tout langage reconnu par une machine de Turing à ruban bi-infini est reconnu par une machine de Turing standard.

Calcul de fonction
On dit qu'un machine de Turing M calcule la fonction f si, sur l'entrée x, M accepte avec $f(x)$ écrit sur son (premier dans le cas de plusieurs rubans) ruban.

Exercice 9.2 *Calculs*
Donner les machines de Turing calculant les fonctions suivantes :

— $f_1 : a^n \mapsto a^n b^n$;

— $f_2 : w \in \{a, b\}^* \mapsto w.\overline{w}$;

— $f_3 : w \in \{a, b\}^* \mapsto w.w$;

— $f_4 : (n, m) \mapsto n - m$ où n, m sont des entiers codés en binaire.

Exercice 9.3 *Composition*
Soient une machine de Turing M_f calculant la fonction f et une machine de Turing M_g calculant la foncction g. Donner une machine de Turing calculant $f \circ g$.

Exercice 9.4 *Composition et reconnaissance*
Soient M_L une machine de Turing reconnaissant le langage $L \subseteq \{a, b\}^*$ et M_f une machine de Turing calculant la fonction $f : \{a, b\}^* \to \{a, b\}^*$.

— Donner une machine de Turing qui reconnaît $f^{-1}(L) = \{x | f(x) \in L\}$;

— Donner une machine de Turing qui reconnaît $f(L) = \{y | \exists x \in L. f(x) = y\}$.

10 TD 10 – Décidabilité

Exercice 10.1 *Questions existentielles*
Rappelez ce que veut dire décidable, non décidable, semi-décidable.
L'un des deux langages suivants est décidable, lequel ? Justifiez.

— $L_1 = \{n |$ les décimales de π contiennent la séquence $1^n\}$;

— $L_2 = \{n |$ les décimales de π contiennent une sous séquence maximale de 1 de longueur $n\}$.

Exercice 10.2 *Clôtures*
Soient L_1 et L_2 des langages décidables, montrer que

— $L_1 \cup L_2$ est décidable ;

— $L_1 \cap L_2$ est décidable ;

— $\overline{L}_1$ est décidable.

Exercice 10.3 *Arrêt universel*
Donner une manière d'encoder une machine de Turing sur un ou plusieurs rubans.
Soit L_U le langage des codes de machines de Turing qui s'arrêtent sur toute entrée.

$$L_U = \{\langle M \rangle \mid \forall w \in \Sigma^* \ M \text{ s'arrête sur } w\}$$

Montrer que L_U est indécidable.

Exercice 10.4 *Arrêt existentiel*
Soit L_E le langage des codes de machines de Turing qui s'arrêtent sur au moins une entrée.

$$L_E = \{\langle M \rangle \mid \exists w \in \Sigma^* \ M \text{ s'arrête sur } w\}$$

Montrer que L_E est indécidable.

Exercice 10.5 *Arrêt en temps n*
Soit L_B le langage des codes de machines de Turing qui s'arrêtent sur une certaine entrée en temps n.

$$L_B = \{\langle M, w, n \rangle \mid \ M \text{ s'arrête sur } w \text{ en au plus } n \text{ étapes}\}$$

L_B est-il décidable ?

11 TD 11 – Thèse de Church et reconnaissance

Exercice 11.1 *Programmation*
Soient f_1, f_2, f_3 des fonctions calculables par une machine de Turing. Montrer que les fonctions suivantes sont calculable par une machine de Turing :

— $x \mapsto$`if` $f_1(x) = 1$ `then` $f_2(x)$ `else` $f_3(x)$

— $x \mapsto$ `while`$(f_1(x) = 1)\{x = f_2(x)\}$`return` $f_3(x)$

Exercice 11.2 *Automate à pile*
Montrer que tout langage reconnaissable par un automate à pile (pas forcément déterministe), est reconnaissable par une machine de Turing.

Exercice 11.3 *Automates à deux piles*
Montrer que tout langage reconnaissable par un automate à deux piles est reconnaissable par une machine de Turing. *Attention, l'automate n'est pas forcément déterministe.*

Exercice 11.4 *Automate à deux piles - bis*
Montrer que tout langage reconnaissable par une machine de Turing est reconnaissable par un automate à deux piles.

Exercice 11.5 *Décidabilité*
Soit $\langle\cdot\rangle$ un encodage des automates. Montrer que les langages suivants sont décidables :

$$A_{AFD} = \{\langle B\rangle\#w \mid B \text{ est un AFD reconnaissant } w\}$$

$$EQ_{AFD} = \{\langle B\rangle\#\langle C\rangle \mid \mathcal{L}(B) = \mathcal{L}(C)\}$$