MPI
MPI*

Lou Chalmain
Galatée Hémery

PRÉPAS SCIENCES

COLLECTION DIRIGÉE PAR **BERTRAND HAUCHECORNE**

INFORMATIQUE

- Cours complet et détaillé
- Nombreux exemples
- Vrai/Faux
- Exercices avec indications
- Corrections et commentaires

ellipses

PRÉPAS SCIENCES
collection dirigée par Bertrand Hauchecorne

Informatique
MPI/MPI*

Lou Chalmain
Agrégée de mathématiques,
professeure d'informatique en MP2I
au lycée privé Fénelon Sainte-Marie (Paris)

Galatée Hémery
Agrégée de mathématiques,
professeure d'informatique en MPI
au lycée privé Fénelon Sainte-Marie (Paris)

ellipses

COLLECTION
PRÉPAS SCIENCES

Retrouvez tous les titres de la collection et des extraits sur www.editions-ellipses.fr

Les notices culturelles « Un scientifique » et « Un peu d'histoire »
des pages de titre des chapitres ont été rédigées par Bertrand Hauchecorne.

Les macros de cet ouvrage ont été réalisées par Nicolas Nguyen en LaTeX.

ISBN 9782340-114449
Dépôt légal : mars 2026

www.editions-ellipses.fr

Un immense merci à
Brigitte Chalmain, Pierre Le Scornet,
Thibault Lestienne et José Lorgeré
pour leurs précieuses relectures.

Avant-propos

La classe préparatoire **MPI** est destinée avant tout à former des étudiants en informatique en leur donnant des bases solides couvrant principalement des aspects théoriques de la discipline mais également des aspects plus techniques comme la programmation.
L'ouvrage de la collection *Prépas sciences* que vous avez entre les mains est spécifique à cet enseignement et vous permettra d'être à l'aise dans cette formation.
En effet, réussir en classes préparatoires nécessite d'assimiler rapidement un grand nombre de connaissances, mais surtout de savoir les utiliser, à bon escient, et les rendre opérationnelles au moment opportun. Bien sûr, l'apprentissage du cours de votre professeur jour après jour est indispensable. Cependant, on constate que pour beaucoup, c'est loin d'être suffisant. Combien d'entre vous ont bien appris leur cours et pourtant se trouvent démunis lors d'un devoir, et plus grave, le jour du concours ? Cette collection a été conçue pour répondre à cette difficulté.

Les chapitres thématiques comportent de nombreux algorithmes, le plus souvent écrits en C ou en OCaml, et vous permettent de développer vos compétences de programmation dans ces deux langages ; leur contenu suit strictement le programme et leur structure est identique.

- **Le cours complet** rassemble tous les résultats à savoir. Il vous permet d'accéder à une connaissance approfondie des notions. Sa relecture est indispensable avant un devoir en classe et *a fortiori* avant l'examen. Les nombreuses situations données en exemple, en particulier les programmes, vous initient aux techniques utiles pour résoudre les exercices classiques. Elles éclairent et illustrent le cours.
- **La partie « vrai/faux »** vous permet de tester votre recul par rapport au programme, vous révèle les mauvais réflexes à corriger et vous met en garde contre les erreurs classiques.
- **Les exercices** sont incontournables pour assimiler le programme et pour répondre aux exigences du concours.
- **Les corrigés** sont rédigés avec soin et de manière exhaustive.

Ainsi l'ouvrage d'informatique, comme ceux de maths et de physique-chimie, vous accompagneront tout au long de l'année et vous guideront dans votre cheminement vers la réussite aux concours.

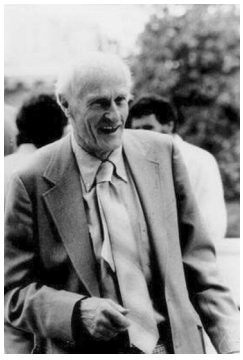
Bertrand Hauchecorne

Sommaire

Chapitre 1

Mots, langages et expressions régulières

UN SCIENTIFIQUE



Stephen Kleene (1909-1984)

Le mathématicien et logicien américain Stephen Kleene étudie à l'université de Princeton où il obtient son doctorat en 1934, sous la direction de son compatriote Alonzo Church.

Ses premiers travaux, élaborés dans les années 1930, concernent la logique formelle ; il introduit en particulier la notion de calculabilité et étudie les fonctions récursives, ce qui permet de préciser le concept d'algorithme. On peut dire que Kleene a posé les bases de l'informatique théorique.

Un peu d'histoire

Leibniz avait imaginé un langage formel pouvant décrire à la fois les mathématiques et plus généralement les sciences mais également des concepts métaphysiques. Ceci devait déboucher sur le *calculus ratiocinator*, machine théorique idéale sachant résoudre tous les problèmes de manière algorithmique.

À la fin du XIX^{e} siècle le logicien allemand Gottlob Frege introduit un langage formel, c'est-à-dire dans lequel on ne s'intéresse qu'à la structure et non au sens, dans le but d'y formaliser l'arithmétique.

Par la suite, plusieurs logiciens comme le Norvégien Axel Thue et l'Américain Emil Post poursuivent des recherches sur ce thème. Le linguiste Noam Chomsky s'en inspire pour ses travaux sur les langues naturelles tandis que des informaticiens s'en emparent pour créer les premiers langages de programmation à l'exemple de John Backus, à l'initiative du FORTRAN.

OBJECTIFS

- Savoir raisonner autour des mots et des langages :
  - Maîtriser le vocabulaire
  - Savoir manipuler les opérations régulières sur les langages
  - Reconnaître, interpréter, analyser et manipuler les expressions régulières
  - Démontrer des propriétés sur les langages réguliers

Cours

Mots sur un alphabet

Définition : *Un **alphabet** est un ensemble fini dont les éléments sont appelés **lettres**.*
On utilise généralement la notation Σ pour cet ensemble, parfois A. Dans ce chapitre, on utilisera uniquement Σ.

Définition : *Un **mot** sur l'alphabet Σ est une suite finie de lettres dans Σ.*
*La **longueur** du mot est le nombre de lettres de la suite. Pour un mot m, on note $|m|$ sa longueur.*

Notations : *On note Σ^* l'ensemble de tous les mots sur l'alphabet Σ.*
Le mot vide est noté ε, il correspond à la suite vide, n'utilisant aucune lettre.

Exemples : $\Sigma_1 = \{a, b, c\}$ est un alphabet à trois lettres. *aaa* et *bac* sont des mots de longueur 3 sur Σ_1, mais *dab* n'est pas un mot sur Σ_1 car $d \notin \Sigma_1$.
$\Sigma_2 = \{(,)\}$ est un alphabet à deux lettres. ()(()) et)() sont des mots sur Σ_2.

⚠ **Remarque :** ε est un mot, mais pas une lettre. Autrement dit, $\varepsilon \in \Sigma^*$ et $\varepsilon \notin \Sigma$.

Notations : *L'ensemble des mots non vides est généralement noté Σ^+.*
Pour un mot m et une lettre $a \in \Sigma$, $|m|_a$ désigne souvent le nombre d'occurrences de la lettre a dans m.

Définition : *Sur l'ensemble des mots, on définit une loi de concaténation.*
*Soit $m = m_1...m_n$ et $s = s_1...s_p$ deux mots. La **concaténation** de m et s, notée ms ou $m \cdot s$, est le mot $ms = m_1...m_n s_1...s_p$.*

Notation : *On note m^q la concaténation de $q \in \mathbb{N}$ fois le mot m.*

Exemples : a^5 est une autre notation pour le mot *aaaaa*.
$(ba)^2(ab)^2$ est une autre notation pour le mot *babaabab*.

Remarque : La loi de concaténation est une loi associative, admettant un élément neutre ε, mais elle n'est pas commutative. On dit que $(\Sigma^*, \cdot)$ est un monoïde *(information pour la culture générale informatique et mathématique).*

Définition : *Soit u, v deux mots sur un alphabet Σ. On dit que :*

- *u est un **préfixe** de v lorsqu'il existe un mot t tel que $ut = v$;*
- *u est un **suffixe** de v lorsqu'il existe un mot t tel que $tu = v$;*
- *u est un **facteur** de v lorsqu'il existe deux mots t, s tels que $tus = v$.*

*Un préfixe, un suffixe ou un facteur u de v est dit **propre** lorsque $u \neq v$.*

Exemples : *abc* est un préfixe et un suffixe de *abc*, mais ni un préfixe propre, ni un suffixe propre. C'est également un facteur de *abc*, mais pas un facteur propre.
ab est un préfixe propre de *abc*, c'est également un facteur propre.
bc est un suffixe propre de *abc*, c'est également un facteur propre.
b est un facteur propre de *abc*, mais ni un préfixe, ni un suffixe.
ε est préfixe, suffixe et facteur de tout mot.

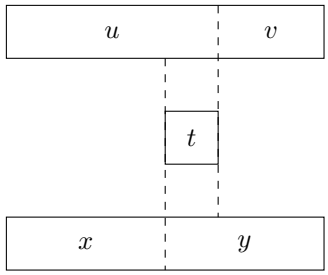
Définition : *Un **sous-mot** de $m = m_1...m_n$ est un mot formé des lettres de m en respectant l'ordre : $s = m_{\phi(1)}...m_{\phi(p)}$ avec ϕ strictement croissante et en particulier $p \leqslant n$.*

Exemple : Le mot *tir* est un sous-mot du mot *aleatoire*.

Remarque : On peut remarquer qu'un facteur d'un mot w est un sous-mot de w particulier dans lequel les lettres sont consécutives dans le mot w.

Lemme 1.1 (Levi).— Soit x, y, u, v des mots sur un alphabet Σ tels que $uv = xy$.
Il existe un unique $t \in \Sigma^*$ tel que ($u = xt$ et $y = tv$) ou bien ($x = ut$ et $v = ty$).

Le schéma suivant illustre un des deux cas du lemme de Levi, qui n'est pas au programme de MPI mais dont la connaissance se révèle souvent bien utile.



■ Langages et opérations régulières

Définition : *Un **langage** est un ensemble de mots sur un alphabet.*

Remarque : $\mathcal{P}(\Sigma^*)$ est l'ensemble des langages sur l'alphabet Σ.

Notation : *On peut noter ε le langage $\{\varepsilon\}$ réduit au mot vide ε. Le langage vide est noté $\emptyset$.*
En particulier, $\varepsilon \neq \emptyset$.

Exemple : $L = \{m \in \{a, b\}^* \mid |m|_a = 2\}$ est le langage des mots sur l'alphabet $\{a, b\}$ contenant exactement deux a.

Définitions à connaître : *Sur les langages, on définit des **opérations régulières** :*

- *l'**union** de deux langages :*

$$L_1 \cup L_2 = \{m \mid m \in L_1 \text{ ou } m \in L_2\}$$

- *la **concaténation** de deux langages, notée L_1L_2 ou bien $L_1 \cdot L_2$:*

$$L_1 \cdot L_2 = \{m_1m_2 \mid m_1 \in L_1, m_2 \in L_2\}$$

- *l'**étoile de Kleene** sur un langage :*

$$L^* = \bigcup_{k \in \mathbb{N}} L^k$$

Notation : *On note $L^n = \underbrace{L...L}_{n \text{ fois}}$ la concaténation de n fois le langage L pour $n \in \mathbb{N}$ et L un langage.*

Remarque : Cette notation impose que $L^0 = \{\varepsilon\}$.
En particulier, pour L le langage vide, on obtient $\emptyset^0 = \varepsilon$ donc $\emptyset^* = \{\varepsilon\}$.

■ Expressions régulières

Définition à connaître : *Soit Σ un alphabet.*
L'ensemble des ***expressions régulières*** *sur Σ est défini par induction :*

- *$\emptyset$ et ε sont des expressions régulières ;*
- *pour tout $a \in \Sigma$, a est une expression régulière ;*
- *si e_1, e_2 sont deux expressions régulières, alors $e_1|e_2$, $e_1 \cdot e_2$ (également noté e_1e_2), et e_1^* sont des expressions régulières.*

Remarque : On utilise des parenthèses comme pour les expressions arithmétiques ou les formules de la logique propositionnelle pour définir les priorités d'opérations et on peut utiliser la convention suivante : *l'étoile de Kleene $*$ est prioritaire sur la concaténation $\cdot$ qui est prioritaire sur l'union $|$.*

Définition : *Un* ***langage régulier*** *est un langage dénoté par une expression régulière selon les règles suivantes :*

- *$\emptyset$ dénote le langage vide et ε dénote le langage réduit au mot vide $\{\varepsilon\}$;*
- *pour tout $a \in \Sigma$, a dénote le langage $\{a\}$ contenant uniquement le mot a de longueur 1 ;*
- *si e_1, e_2 sont deux expressions régulières dénotant des langages L_1 et L_2, alors*
  - *$e_1|e_2$ dénote $L_1 \cup L_2$;*
  - *e_1e_2 (ou $e_1 \cdot e_2$) dénote $L_1L_2 = L_1 \cdot L_2$;*
  - *e_1^* dénote L_1^*.*

Exemple : b^*aba^* est une expression régulière qui dénote le langage L des mots commençant par des b, puis comportant le facteur ab, puis des a.
Les mots de ce langage L peuvent s'écrire sous la forme $b^{n_1}aba^{n_2}$ avec n_1 et n_2 des entiers naturels.

Remarque : Il existe des langages non réguliers. Par exemple, le langage $L = \{a^nb^n \mid n \geqslant 0\}$ n'est pas régulier.

⚠ **Remarque :** Dans la littérature, « langage rationnel » est synonyme de « langage régulier ». De même, on peut dire « expression rationnelle » à la place de « expression régulière ». L'adjectif utilisé dans le programme de MPI est « régulier ».

Remarque : À l'aide des expressions régulières, on peut dénoter formellement des motifs.

Le programme de MP2I s'intéresse aux algorithmes de recherche de motifs dans un texte, en particulier avec les algorithmes de Rabin-Karp et de Boyer-Moore, dans le cas où les motifs recherchés sont des mots précis. On peut généraliser ce problème avec la recherche de motifs ayant une forme particulière. Par exemple, il peut s'agir de rechercher toutes les adresses mails valides dans un texte.
Les expressions régulières permettent de formaliser la spécification des motifs à chercher dans une chaîne de caractères. On trouve de nombreuses applications :

- la recherche de motifs dans un texte, et de manière plus avancée la recherche d'un ensemble de mots vérifiant un certain nombre de propriétés ;
- en bio-informatique : l'étude du génome comme des séquences sur un alphabet fini (A,T,C,G), la recherche d'un gène particulier dans une séquence ;
- dans la compilation d'un programme : la transformation d'un fichier contenant du texte en une série d'instructions exécutables par l'ordinateur, l'identification des mots clefs du langage de programmation, des nombres, des noms de variables, etc.

Cours

Remarque : Une expression régulière dénote un langage **unique** mais un langage peut être dénoté par plusieurs expressions régulières. Par exemple $(a \cdot b)|(a \cdot a)$ et $a \cdot (a|b)$ sont deux expressions régulières qui dénotent le langage $\{ab, aa\}$.

On peut donner une définition équivalente des langages réguliers qui découle de la définition inductive des expressions régulières.

Définition : *L'ensemble $R(\Sigma)$ des langages réguliers sur Σ est* ***la plus petite partie*** *de $\mathcal{P}(\Sigma^*)$ au sens de l'inclusion contenant $\emptyset, \varepsilon, \{a\}$ pour tout $a \in \Sigma$ et stable par* ***union finie, concaténation finie*** *et* ***étoile de Kleene****.*

Proposition 1.2.— Soit L un langage régulier dénoté par une expression régulière e. Si $L \neq \emptyset$, alors il existe une expression régulière e' n'utilisant pas le symbole $\emptyset$ et dénotant L.

Démonstration

On procède par induction structurelle sur l'expression régulière e dénotant L.

Cas de base :

- si $e = \varepsilon$ ou $a \in \Sigma$, l'expression régulière n'utilise pas $\emptyset$;
- si $e = \emptyset$, le langage est vide.

Cas inductifs :

- si $e = e_1|e_2$ avec e_1 et e_2 dénotant L_1 et L_2 vérifiant tous les deux la propriété :
  - si $L_1 = L_2 = \emptyset$, alors le langage dénoté est vide ;
  - si $L_1 = \emptyset$ et $L_2 \neq \emptyset$, par hypothèse d'induction, L_2 est dénoté par une expression régulière e_2' sans $\emptyset$ et $L_2 = L$;
  - la situation est symétrique lorsque $L_1 \neq \emptyset$ et $L_2 = \emptyset$;
  - si les deux langages sont non vides, par hypothèse d'induction, L_1 et L_2 sont respectivement dénotés par des expressions régulières e_1' et e_2' et $e' = e_1'|e_2'$ dénote L.
- si $e = e_1 \cdot e_2$ avec e_1 et e_2 dénotant L_1 et L_2 vérifiant tous les deux la propriété :
  - si $L_1 = \emptyset$ ou $L_2 = \emptyset$, alors $L = \emptyset$;
  - sinon, on applique de même l'hypothèse d'induction pour obtenir e_1' et e_2' et $e' = e_1' \cdot e_2'$ dénote L.
- si $e = e_1^*$ avec e_1 qui dénote L_1 vérifiant la propriété :
  - si $L_1 = \emptyset$, alors $e' = \varepsilon$ dénote $L = \{\varepsilon\}$;
  - sinon, on applique de même l'hypothèse d'induction pour obtenir e_1' et $e' = (e_1')^*$ dénote L.

On peut donc conclure par le principe d'induction structurelle. ▲

Remarque : Cette démonstration constitue un exemple important de démonstration par induction structurelle sur les expressions régulières et peut servir de modèle pour les exercices.

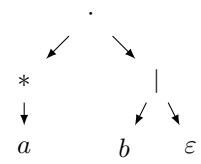
Notation : *On peut utiliser Σ dans une expression régulière pour dénoter le langage formé des mots de longueur 1.*

Exemple : Si $\Sigma = \{a, b, c\}$, l'expression régulière $\Sigma^* abc$ est une abréviation pour $(a|b|c)^* abc$ et $a\Sigma$ dénote le langage des mots de longueur 2 commençant par un a.

■ Représentation des expressions régulières

Pour représenter une expression régulière, on peut utiliser des **arbres** dont la racine et les noeuds internes sont des opérations $|, ^*, \cdot$ et les feuilles sont des expressions régulières de base $\emptyset, \varepsilon, a$ ($a \in \Sigma$). Ces arbres participent à l'analyse syntaxique de l'expression et précisent les priorités sans utiliser de parenthèses.

Exemple : $(a^*) \cdot (b|\varepsilon)$ est représentée par



En OCaml, il est assez naturel de construire un type pour les expressions régulières en s'appuyant sur la définition inductive :

```
1  type regexp =
2      | Epsilon
3      | Vide
4      | Lettre of char
5      | Union of regexp * regexp
6      | Concat of regexp * regexp
7      | Kleene of regexp
8
9  let e =
10     Union ( Kleene (Lettre 'a'),
11             Concat (Lettre 'a', Lettre 'b')
12           )
```

Code 1.1 – Langage OCaml

En C, on peut utiliser un type structuré proche des arbres.

```
1  struct RegExp {
2      char etiquette;
3      struct RegExp *filsd;
4      struct RegExp *filsg;
5  };
6  typedef struct RegExp RegExp_t;
7
8  int main(void) {
9      [...]
10     RegExp_t a = {.etiquette = 'a',.filsd = NULL , .filsg = NULL};
11     RegExp_t b = {.etiquette = 'b',.filsd = NULL , .filsg = NULL};
12     // On peut choisir une lettre pour indiquer le mot vide, ici on choisit E.
13     RegExp_t eps = {.etiquette = 'E',.filsd = NULL , .filsg = NULL};
14     RegExp_t star = {.etiquette = '*',.filsd = &a , .filsg = NULL};
15     RegExp_t cup = {.etiquette = '|',.filsd = &b , .filsg = &eps};
16     RegExp_t e = {.etiquette = '.',.filsd = &star,.filsg = &cup};
17     [...]
18 }
```

Code 1.2 – Langage C

Cours

■ Expressions régulières étendues

Les expressions régulières étendues sont utilisées par certains langages de programmation et certains utilitaires. Notamment, on peut chercher un certain motif dans les fichiers d'un répertoire avec la commande `grep` suivie d'une expression régulière étendue avec un système Linux. S'il existe des conventions de notation très diverses, la norme POSIX est une tentative pour remédier à la multiplication des notations.

Aucune connaissance sur la norme POSIX n'est exigible selon le programme de MPI.

En plus des notations des expressions régulières « basiques » :

Caractère	Description	Exemple
.	n'importe quel caractère	. est équivalent à Σ
$*$	étoile de Kleene	$a*$ pour un nombre quelconque de a
$+$	répétition avec au moins une occurrence d'un motif	$a+$ pour un a ou plus
^	indique que le motif doit être cherché en début de ligne	`grep ^a` cherche les lignes commençant par un a
$	indique que le motif doit être cherché en fin de ligne	`grep $a` cherche les lignes se terminant par un a
[...]	liste un choix parmi plusieurs caractères	$[aeiou]$ pour n'importe quelle voyelle
[^...]	liste plusieurs caractères interdits	$[^aeiou]$ pour n'importe quelle consonne
$[x-y]$	intervalle de caractères	$[a-d]$ est équivalent à $[abcd]$
(...)	délimite un groupe	$ab(.b)*ba$, l'étoile concerne $.b$
?	caractère optionnel	? est équivalent à $\Sigma\|\varepsilon$
(...)?	motif optionnel	$aa(ab)?$ correspond au langage $\{aaab, aa\}$
...{n}	répétition n fois	$a\{5\}$ correspond à $aaaaa$
...{n,}	répétition n fois ou plus	$a\{5,\}$ correspond à $aaaaaa*$
...{,n}	répétition n fois ou moins	$a\{,4\}$ correspond à $\varepsilon\|a\|aa\|aaa\|aaaa$
...{n,m}	répétition entre n et m fois	$a\{3,5\}$ correspond à $aaa(\varepsilon\|a\|aa)$

Les caractères ayant un sens particulier (^, [, ., $, $*$, (,), +, |, ?, <, >) doivent être précédés d'un caractère d'échappement \ pour être utilisés et `\n` désigne un passage à la ligne.
Il existe également d'autres outils moins courants pour les expressions régulières étendues.

Vrai/Faux

	Vrai	Faux
1. Le mot ε est de longueur 1.	☐	☐
2. Le mot $baab$ a deux facteurs propres distincts.	☐	☐
3. Le mot bb est un sous-mot de $baab$.	☐	☐
4. Les expressions régulières $(a \cdot b)^*$ et $(a^*) \cdot (b^*)$ dénotent le même langage.	☐	☐
5. $\emptyset \cdot ((aa\|bb)^*) \cdot (c^*)$ dénote le langage vide.	☐	☐
6. Le langage des mots sur $\Sigma = \{a, b, c\}$ contenant un nombre pair de a est régulier.	☐	☐
7. Le langage $\bigcup_{n \in \mathbb{N}} \{a^n b^n\}$ est régulier.	☐	☐
8. Tous les langages finis sont réguliers.	☐	☐
9. $((a\|b) \cdot (a\|b))^*$ dénote le langage des mots de longueur paire sur $\Sigma = \{a, b\}$.	☐	☐
10. Pour tout langage régulier, il existe une infinité d'expressions régulières le dénotant.	☐	☐

Énoncé des exercices

Exercice 1.1 : Pour chacune des expressions suivantes, dire s'il s'agit d'une expression régulière sur l'alphabet $\{a, b\}$. Lorsque c'est le cas, donner un arbre représentant l'expression, ainsi que sa représentation en OCaml.

1. a
2. c
3. $a|b^*$
4. $(a|b)^*$
5. $(abba = baba)^*|bb$
6. abb

Exercice 1.2 : On se place pour cet exercice dans le cas où $\Sigma = \{a, b\}$.
Pour chacun des langages suivants, proposer une expression régulière le dénotant.

1. L_1 le langage des mots commençant par ab.
2. L_2 le langage des mots terminant par bab.
3. L_3 le langage des mots admettant comme facteur aba.
4. L_4 le langage des mots admettant aa comme préfixe et aaa comme suffixe.
5. L_5 le langage des mots n'admettant pas bba comme préfixe.
6. L_6 le langage des mots n'admettant pas aa comme facteur.
7. L_7 le langage des mots contenant un nombre pair de fois la lettre a.
8. L_8 le langage des mots de la forme w tel que le nombre de b de w, noté $|w|_b$ vérifie $|w|_b \equiv 2[3]$.
9. L_9 le langage des mots de longueur impaire.

Exercice 1.3 : Décrire avec une phrase les langages dénotés par les expressions suivantes sur l'alphabet $\{a, b\}$.

1. $(aa|bb)(a|b)^*$
2. $b^*(a|\varepsilon)$
3. $((ab|bb)^*(b|abba))^*\emptyset((ba^*b)^*b)^*$
4. $b^*(ab^*ab^*)^*ab^*$
5. $((ab)^*|(ba)^*)\emptyset^*$

Exercice 1.4 : Dans cet exercice, on s'intéresse aux différentes expressions régulières dénotant le même langage. Pour cela, on montre quelques égalités sur les langages.

1. Montrer que pour L un langage quelconque, $L^* = (L^*)^*$ et en déduire que pour e une expression régulière, e^* et $(e^*)^*$ dénotent le même langage.
2. Montrer que pour L, M deux langages quelconques, $(L^* \cdot M)^* = \{\varepsilon\} \cup ((L \cup M)^* \cdot M)$ et en déduire une propriété sur les expressions régulières.

Exercice 1.5 : Soit L un langage non vide. Montrer que $\varepsilon \in L$ si et seulement si $L \subset L^2$.

Exercice 1.6* : Soit x, y, z des mots sur un alphabet Σ tels que $xy = yz$ et $x \neq \varepsilon$.
En utilisant le lemme de Levi, montrer qu'il existe deux mots u, v et un entier k tels que

$$x = uv, \quad y = (uv)^k u = u(vu)^k \quad \text{et} \quad z = vu$$

Exercice 1.7* : Soit x, y des mots sur un alphabet Σ tels que $xy = yx$.
En utilisant le lemme de Levi, montrer qu'il existe un mot u et deux entiers i, j tels que $x = u^i$ et $y = u^j$.

Exercice 1.8* : Soit L un langage sur un alphabet Σ et $u \in \Sigma^*$.
Montrer que si L est régulier, alors $u^{-1}L$ le langage défini par $u^{-1}L = \{w \in \Sigma^* \mid uw \in L\}$ est également régulier. Ce langage est appelé ***résiduel*** de L par rapport à u..

Exercice 1.9 (Équations de mots)*** :** Soit Σ un alphabet fini et x une variable de type mot.
Étant donné $w \in \Sigma^*$ et $1 \leqslant i \leqslant j \leqslant |w|$, on pose $w[i : j] = w_i w_{i+1} ... w_j$ le facteur de w entre les indices i et j (inclus). On pourra aussi utiliser la notation $w[i :]$ pour signifier $w[i : |w|]$ et $w[: j]$ pour $w[1 : j]$. On conviendra que $w[1 : 0] = \varepsilon$.
Une équation de mots à une variable est une équation de la forme

$$(1) : \quad A_0 x A_1 x ... A_r = B_0 x B_1 x ... x B_s$$

où $A_0, ..., A_r, B_0, ..., B_s \in \Sigma^*$ sont des mots. Une solution est un mot $x \in \Sigma^*$ qui satisfait (1).
Par exemple, sur l'alphabet $\Sigma = \{a, b\}$, l'équation $xxbaababa = ababaxabx$ admet $x = ababaababa$ comme solution mais pas $x = a$.
1. Déterminer toutes les solutions de $abx = xba$ sur l'alphabet $\Sigma = \{a, b\}$.
Proposer une expression régulière dénotant le langage solution.
2. Soit $U, V \in \Sigma^*$ deux mots non vides. Montrer que l'équation $xx = UxV$ admet au plus une solution que l'on identifiera ainsi qu'une condition nécessaire et suffisante d'existence.
3. Montrer que le cas général (1) peut toujours se ramener au cas ou $A_0 \neq \varepsilon$ et $B_0 = \varepsilon$.
4. On suppose maintenant que l'équation est de la forme obtenue ci-dessus, c'est-à-dire $A_0 \neq \varepsilon$ et

$$(2) : \quad A_0 x A_1 x ... A_r = x B_1 x ... x B_s$$

Montrer que si x est solution de (2) alors il existe $k \in \mathbb{N}$ et $0 \leqslant l < |A_0|$ tels que $x = A_0^k A_0[1 : l]$.
5. On dit que $A, B \in \Sigma^*$ sont *conjugués* lorsqu'il existe $u, v \in \Sigma^*$ tels que $A = uv$ et $B = vu$.
On dit que $P \in \Sigma^*$ est *primitif* lorsque $P \neq \varepsilon$ et P n'apparaît que deux fois comme facteur de PP.
Formellement, P est primitif si et seulement si pour tout $u, v \in \Sigma^*$, $P^2 = uPv$ implique que $u = \varepsilon$ ou $v = \varepsilon$.
Par exemple, ab est primitif mais $P = abab = (ab)^2$ ne l'est pas car il apparaît trois fois dans $P^2 = (ab)^4$.
Soit $R \in \Sigma^*$ primitif et $k \geqslant 1$. Montrer que l'ensemble des solutions de $R^k x = x R^k$ est $\{R\}^*$.
6. Montrer que pour tout $P \in \Sigma^*$ non vide, il existe $R \in \Sigma^*$ primitif et $k \in \mathbb{N}$ tel que $P = R^k$.
7. Soit P primitif et $k \geqslant 1$. Donner l'ensemble des $x, y \in \Sigma^*$ tels que $xPy = P^k$.
8. Que peut-on dire de l'ensemble des solutions de (2) dans le cas où $r \neq s$.
9. Résoudre (2) dans le cas $r = s = 1$ et $A_1 = \varepsilon$. On identifiera une condition nécessaire et suffisante d'existence.
10. Résoudre (2) dans le cas $r = s = 2$ et $A_2 = \varepsilon$.

D'après un oral du concours ENS 2022, MP

Indications

Ex. 1.4

On peut montrer les égalités par double inclusion en reprenant les définitions des opérations sur les langages.

Ex. 1.6

Montrer le résultat par récurrence forte sur $|y| + |x| = |y| + |z|$.

Ex. 1.7

Montrer le résultat par récurrence forte sur $|x| + |y|$.

Ex. 1.8

On pourra procéder par récurrence sur la longueur de u et pour u de longueur 1 par induction.

Ex. 1.9

1. On pourra procéder par récurrence sur la longueur du mot pour justifier proprement.
4. Raisonner sur la longueur de la solution x.
5. On peut utiliser la question précédente, injecter la solution dans l'équation et en déduire de nouvelles propriétés en exploitant le caractère primitif.
6. On peut raisonner par récurrence sur $|P|$ et exploiter les questions **2.** et **5.**.

Corrigé des vrai/faux

1	2	3	4	5	6	7	8	9	10
F	F	V	F	V	V	F	V	V	V

Quelques explications

1. Le mot ε est le mot vide, il est de longueur nulle.

2. Les facteurs propres de $baab$ sont $b, ba, baa, ab, aab, a, aa, \varepsilon$. Il ne faut pas oublier les préfixes et suffixes propres et le mot vide.

3. Pour construire bb avec $baab$, on peut utiliser la première et la dernière lettre.

4. Non, $abab$ est dans le langage dénoté par $(a \cdot b)^*$, mais pas de la forme $a^n b^m$ comme imposé par la deuxième expression.

5. $\emptyset$ dénote le langage vide et tout mot du langage dénoté commence par un mot de ce langage, or il n'en existe pas.

6. L'expression régulière $((b|c)^*a(b|c)^*a(b|c)^*)^*$ convient.

7. Il s'agit de $\{a^n b^n \mid n \geqslant 0\}$ qui est l'exemple de langage non régulier donné. En particulier, une union dénombrable de langages réguliers n'est pas nécessairement un langage régulier.

8. Pour tout mot m, il existe une expression régulière, qui s'écrit comme le mot m avec les mêmes lettres qui dénote le langage $\{m\}$.
Pour un langage fini $L = \{m_1, \ldots, m_n\}$, l'expression régulière $m_1|m_2|\ldots|m_n$ dénote L.

9. Les lettres sont rassemblées par paquets de deux.

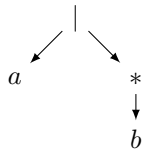
10. Pour L un langage régulier dénoté par e, les expressions régulières $\underbrace{e|\ldots|e|e}_{n \text{ fois}}$ dénotent L pour $n \in \mathbb{N}^*$. Il en existe une infinité.

Corrigé

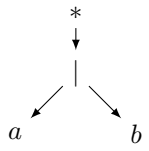
Corrigé des exercices

Exercice 1.1

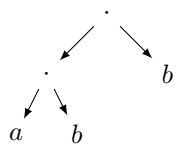
1. a est bien une expression régulière pour $\{a, b\}$.
L'arbre feuille a la représente.
En OCaml, on a la représentation : `Lettre 'a'`.
2. c n'est pas une expression régulière pour $\{a, b\}$ car c n'est pas une lettre de l'alphabet.
3. $a|b^*$ est bien une expression régulière pour $\{a, b\}$.
L'arbre suivant en est une représentation, car le connecteur principal est | (priorité sur l'étoile) :



En OCaml, on a la représentation :
`Union (Lettre 'a', Kleene (Lettre 'b'))`.
4. $(a|b)^*$ est bien une expression régulière pour $\{a, b\}$.
L'arbre suivant en est une représentation, car le connecteur principal est | (priorité sur l'étoile) :



En OCaml, on a la représentation :
`Kleene (Union (Lettre 'a', Lettre 'b'))`.
5. $(abba = baba)^*|bb$ n'est pas une expression régulière pour $\{a, b\}$ car $=$ ne doit pas apparaître.
6. abb est bien une expression régulière pour $\{a, b\}$.
L'arbre suivant en est une représentation, mais ce n'est pas le seul :



En OCaml, on a la représentation :
`Concat (Concat (Lettre 'a', Lettre 'b'), Lettre 'b')`.

Exercice 1.2

On notera $\mathcal{L}(e)$ le langage dénoté par une expression régulière e.

1. $L_1 = \mathcal{L}(ab(a|b)^*)$

2. $L_2 = \mathcal{L}((a|b)^*bab)$

3. $L_3 = \mathcal{L}((a|b)^*aba(a|b)^*)$

4. $L_4 = \mathcal{L}(aa(a|b)^*aaa|aaa|aaaa)$

Attention, il ne faut pas oublier aaa et $aaaa$.

5. $L_5 = \mathcal{L}((a|ba|bbb)(a|b)^*|\varepsilon|b|bb)$

6. $L_6 = \mathcal{L}((b|ab)^*(a|\varepsilon))$

Pour que aa n'apparaisse pas, il faut que chaque a soit suivi d'au moins un b, sauf éventuellement le dernier lorsqu'il est en fin de mot.

7. $L_7 = \mathcal{L}((b^*ab^*ab^*)^*)$

Les a doivent être ajoutés deux par deux.

8. $L_8 = \mathcal{L}((a^*ba^*ba^*ba^*)^*a^*ba^*ba^*)$

9. $L_9 = \mathcal{L}((aa|bb|ab|ba)^*(a|b))$

Exercice 1.3

1. Le langage des mots commençant pas aa ou bb est dénoté par $(aa|bb)(a|b)^*$.

2. Le langage des mots constitués uniquement de b avec éventuellement un a à la fin est dénoté par $b^*(a|\varepsilon)$.

3. L'expression $((ab|bb)^*(b|abba))^*\emptyset((ba^*b)^*b)^*$ dénote le langage vide à cause de la présence de $\emptyset$.

4. Le langage des mots contenant un nombre impair de a est dénoté par $b^*(ab^*ab^*)^*ab^*$.

5. Le langage des mots de longueur paire alternant les a et les b est dénoté par $((ab)^*|(ba)^*)\emptyset^*$. On remarque que $\emptyset^*$ dénote le même langage que ε.

Exercice 1.4

1. Montrons le résultat par double inclusion.

Première inclusion :

Par définition,

$$(L^*)^* = \bigcup_{k\in\mathbb{N}} (L^*)^k$$

Donc en particulier $L^* = (L^*)^1 \subset (L^*)^*$.

Deuxième inclusion :

Soit $w \in (L^*)^*$. Il existe $k \in \mathbb{N}$ tel que $w \in (L^*)^k$.

Si $k = 0$, on a $w = \varepsilon \in L^*$.

Sinon, il existe k mots $w_1, \ldots, w_k$ dans L^* tels que $w = w_1 \cdot \cdots \cdot w_k$.

Par définition de L^*, pour tout i entre 1 et k, il existe n_i tel que $w_i \in L^{n_i}$.

Ainsi pour $k \geqslant 1$, on obtient $w \in L^{n_1} \cdot \cdots \cdot L^{n_k} = L^{n_1+\cdots+n_k} \subset L^*$.

Ceci est vrai pour tout choix de w, donc $(L^*)^* \subset L^*$.

On en déduit l'égalité.

Pour e une expression régulière et L le langage dénoté par e, on a par définition e^* dénote L^* et $(e^*)^*$ dénote $(L^*)^*$.

Par égalité des langages, on obtient que e^* et $(e^*)^*$ dénotent bien le même langage.

Corrigé

2. Montrons le résultat par double inclusion.
Première inclusion :
Soit $w \in (L^* \cdot M)^*$.
Par définition de l'étoile de Kleene, il existe $k \in \mathbb{N}$ tel que $w \in (L^* \cdot M)^k$.
Puis, il existe $w_1, \dots, w_k$ des mots de $L^* \cdot M$ tels que $w = w_1 \dots w_k$.
Puis, de la même façon, chaque w_i s'écrit de la forme $w_i = u_{i,1} \dots u_{i,n_i} v_i$ avec n_i un entier naturel, les $u_{i,j}$ dans L et v_i dans M.
Ainsi, on obtient $w = u_{1,1} \dots u_{1,n_1} v_1 \dots u_{k,1} \dots u_{k,n_k} v_k$.
En particulier, si $k = 0$, $w = \varepsilon \in \{\varepsilon\} \cup ((L \cup M)^* \cdot M)$.
Sinon, on a $u_{1,1} \dots u_{1,n_1} v_1 \dots u_{k,1} \dots u_{k,n_k} \in (L \cup M)^{n_1 + \dots + n_k + (k-1)} \subset (L \cup M)^*$ et $v_k \in M$.
Donc, dans tous les cas, $w \in \{\varepsilon\} \cup ((L \cup M)^* \cdot M)$.
Ceci est vrai pour tout choix de w, donc $(L^* \cdot M)^* \subset \{\varepsilon\} \cup ((L \cup M)^* \cdot M)$.
Deuxième inclusion :
Soit $w \in (L \cup M)^* \cdot M$.
De la même façon que précédemment, w s'écrit de la forme $w = u_1 \dots u_k v$ avec k un entier naturel, les u_i dans $L \cup M$ et $v \in M$.
On note $\{i_1, \dots, i_m\}$ le sous-ensemble d'indices, numérotés de façon croissante, tel que $u_i \in M$.
Si cet ensemble est vide, on a directement $w \in L^* \cdot M \subset (L^* \cdot M)^*$.
Sinon, on a alors $u_1 \dots u_{i_1} \in L^* \cdot M$, puis $u_{i_j+1} \dots u_{i_{j+1}} \in L^* \cdot M$ pour tout $1 \leqslant j \leqslant m$ et enfin $u_{i_m+1} \dots u_k v \in L^* \cdot M$.
Ainsi, $w \in (L^* \cdot M)^*$.
Ceci est vrai pour tout choix de w donc $(L \cup M)^* \cdot M \subset (L^* \cdot M)^*$.
On a de plus $\{\varepsilon\} \subset (L^* \cdot M)^*$ par définition de l'étoile de Kleene.
Donc, au total $\{\varepsilon\} \cup ((L \cup M)^* \cdot M) \subset (L^* \cdot M)^*$.
On en déduit l'égalité.
Pour e et f deux expressions régulières et L et M les langages dénotés par e et f, par définition $(e^* f)^*$ dénote $(L^* \cdot M)^*$ et $\varepsilon | (e|f)^* f$ dénote $\{\varepsilon\} \cup ((L \cup M)^* \cdot M)$.
Par égalité des langages, on obtient que $(e^* f)^*$ et $\varepsilon | (e|f)^* f$ dénotent bien le même langage.

Exercice 1.5

Montrons le résultat par double équivalence.
Supposons que $\varepsilon \in L$. On a alors, pour tout $w \in L$, $\varepsilon \cdot w \in L \cdot L = L^2$.
Ainsi, $L \subset L^2$.
Réciproquement, supposons maintenant que $L \subset L^2$.
On a supposé L non vide, donc L admet un mot w_0 de longueur minimum.
L'inclusion donne l'existence de w_1, w_2 des mots de L tel que $w_0 = w_1 \cdot w_2$.
En considérant la longueur des mots, on a $|w_0| = |w_1| + |w_2| \geqslant 2|w_0|$ car $|w_0|$ est minimum.
Comme $|w_0|$ est positif, on a nécessairement $|w_0| = 0$ pour vérifier cette inégalité et $w_0 = \varepsilon$.
On a donc montré que $\varepsilon \in L$.

Exercice 1.6

Montrons le résultat par récurrence forte sur $|y| + |x| = |y| + |z| \geqslant 1$ (car $xy \neq \varepsilon$).

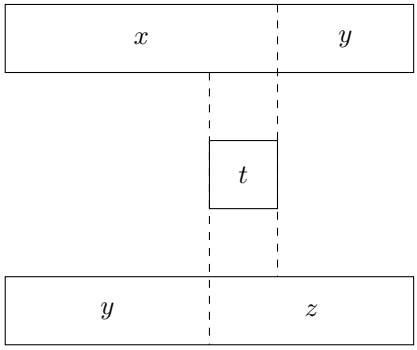
Initialisation :

Pour $|y| + |x| = 1$, nécessairement $|x| = 1$ car x non vide, on remarque que $v = x = z$, $u = y = \varepsilon$ et $k = 0$ permettent d'obtenir le résultat.

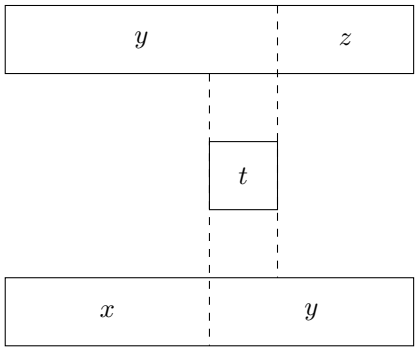
Hérédité :

Dans l'hérédité, on distingue deux cas et on suppose que la proposition est vraie pour tout triplet (x', y', z') de mots tel que $x'y' = y'z'$, $x' \neq \varepsilon$ et $|x'| + |y'| < |x| + |y|$:

— Si $|y| - |x| \leqslant 0$, alors $|x| \geqslant |y|$, d'après le lemme de Levi, on a l'existence de $t \in \Sigma^*$ tel que $x = yt$ et $z = ty$. On peut prendre $u = y$, $v = t$ et $y = (yt)^0 y = y(ty)^0$.



— Si $|y| - |x| > 0$, alors $|x| < |y|$, d'après le lemme de Levi, on a l'existence de $t \in \Sigma^*$ et $t \neq \varepsilon$ tel que $y = xt = tz$.
On a $|x| + |t| < |x| + |y|$ car $x \neq \varepsilon$ donc on peut appliquer l'hypothèse de récurrence.



Ainsi, il existe u, v, k tel que $x = uv$, $t = (uv)^k u = u(vu)^k$ et $z = vu$.
Puis $y = xt = (uv)(uv)^k u = tz = u(vu)^k(vu)$, i.e. $y = (uv)^{k+1}u = u(vu)^{k+1}$.

On en déduit, par récurrence forte, que le résultat est vrai pour tous mots x, y, z tels que $xy = yz$ et $x \neq \varepsilon$.

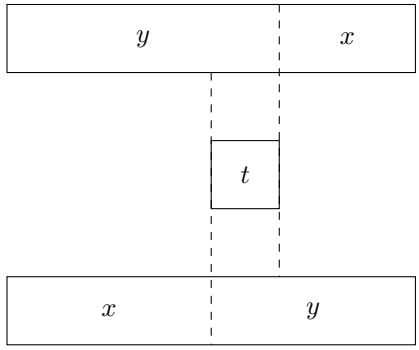
Exercice 1.7

Montrons le résultat par récurrence forte sur $|x| + |y|$.

Initialisation :

Dans le cas $|x| + |y| = 0$, on a nécessairement $x = y = \varepsilon$. Dans ce cas $i = j = 0$ et $u = \varepsilon$ permettent d'obtenir le résultat.

Hérédité :
Soit x, y tel que $xy = yx$. On suppose le résultat vrai pour tous les couples de mots (x', y') tel que $|x'| + |y'| < |x| + |y|$ et $x'y' = y'x'$.
Si $|x| = |y|$, par identification lettre à lettre, on obtient $x = y$. Dans ce cas $u = x$ et $i = j = 1$ permettent d'obtenir le résultat.
Si $x = \varepsilon$, $u = y$ et $i = 0, j = 1$ convient.
Si $y = \varepsilon$, $u = x$ et $i = 1, j = 0$ convient.
Sinon, quitte à échanger x et y, on peut supposer y strictement plus long que x.
D'après le lemme de Levi, il existe t un mot tel que $y = tx = xt$.



Comme x n'est pas réduit à ε, on a $|x| + |t| = |xt| = |xy| - |x| < |x| + |y|$ et on peut appliquer l'hypothèse de récurrence à x et t.
Il existe donc u un mot et i, j des entiers tels que $x = u^i$ et $t = u^j$.
On a alors $y = xt = u^i u^j = u^{i+j}$.
On en déduit le résultat attendu pour x, y.

Exercice 1.8

Montrons d'abord le résultat pour u de longueur 1.
Pour cela, on peut procéder par induction structurelle sur les langages réguliers.
Cas de base :

- Si $L = \{\varepsilon\}$, on a $u^{-1}L = \emptyset$ régulier.
- Si $L = \emptyset$, on a $u^{-1}L = \emptyset$ régulier.
- Si $L = \{a\}$ avec $a \in \Sigma$, on a $u^{-1}L = \emptyset$ régulier si $a \neq u$ et $u^{-1}L = \{\varepsilon\}$ régulier si $a = u$.

Cas inductifs :

- Si $L = L_1 \cup L_2$ avec L_1, L_2 des langages réguliers vérifiant la propriété.
  On a $u^{-1}L = \{w \in \Sigma^* \mid uw \in L_1 \cup L_2\} = \{w \in \Sigma^* \mid uw \in L_1\} \cup \{w \in \Sigma^* \mid uw \in L_2\} = u^{-1}L_1 \cup u^{-1}L_2$.
  Ces deux langages sont réguliers et les langages réguliers sont stables par union par définition donc $u^{-1}L$ est régulier.

- Si $L = L_1 \cdot L_2$ avec L_1, L_2 des langages réguliers vérifiant la propriété.
  On a $u^{-1}L = \{w \in \Sigma^* \mid uw \in L_1 \cdot L_2\}$. Alors en notant $\varepsilon(L_1) = \{\varepsilon\} \cap L_1$, un langage régulier, on a $u^{-1}L = \{w \in \Sigma^* \mid uw \in L_1\} \cdot L_2 \cup \varepsilon(L_1) \cdot \{w \in \Sigma^* \mid uw \in L_2\} = u^{-1}L_1 \cdot L_2 \cup \varepsilon(L_1) \cdot u^{-1}L_2$.
  Ces quatres langages sont réguliers et les langages réguliers sont stables par union et concaténation par définition donc $u^{-1}L$ est régulier.

— Si $L = L_1^*$ avec L_1 un langage régulier vérifiant la propriété.
On a $u^{-1}L = \{w \in \Sigma^* \mid uw \in (L_1)^*\} = \{w \in \Sigma^* \mid uw \in L_1 \cdot (L_1)^*\}$ car $u \neq \varepsilon$.
Puis $u^{-1}L = \{w \in \Sigma^* \mid uw \in L_1\} \cdot L_1^* = u^{-1}(L_1) \cdot L$.
Ces deux langages sont réguliers et les langages réguliers sont stables par concaténation par définition donc $u^{-1}L$ est régulier.

Montrons le résultat général pour tout langage régulier L par récurrence sur $|u|$.
Initialisation :
Pour $|u| = 0$, on a $u = \varepsilon$ et $u^{-1}L = \{w \in \Sigma^* \mid w \in L\} = L$ régulier.
Hérédité :
Soit u de longueur $n \geqslant 1$, on suppose que pour un mot de longueur $n-1$, le résultat est vrai.
On a $u = va$ avec $a \in \Sigma$ et $v \in \Sigma^*$.

$$u^{-1}L = \{w \in \Sigma^* \mid vaw \in L\} = \{w \in \Sigma^* \mid aw \in v^{-1}L\} = a^{-1}(v^{-1}L)$$

Ainsi, par hypothèse de récurrence, $v^{-1}L$ est régulier et grâce à la démonstration par induction pour un mot de longueur 1 (ici a), on obtient $a^{-1}(v^{-1}L)$ régulier et donc $u^{-1}L$ régulier.

Exercice 1.9

1. Montrons par récurrence sur k qu'il n'existe pas de solution de longueur $2k$ et qu'il existe une unique solution de longueur $2k+1$ de la forme $x = (ab)^k a$.
Initialisation :
Pour $k = 0$,

— Si x est de longueur 0, on obtient $ab = ba$ qui est faux, donc il n'y pas de solution de longueur 0.

— Si x est de longueur 1, on obtient $x = a$ par identification, c'est l'unique possibilité.

Hérédité :
Soit $k \geqslant 1$. On suppose le résultat vrai pour le rang $k-1$.
Si x est solution de $abx = xba$ de longueur au moins 2, alors x admet ab comme préfixe, il existe y un mot tel que $x = aby$.
Ainsi, l'équation devient $ababy = abyba$, que l'on simplifie en $aby = yba$.

— Si x est de longueur $2k$, alors y est de longueur $2(k-1)$ et par hypothèse de récurrence, il n'existe pas de y vérifiant $aby = yba$ donc il n'existe pas de solution x de longueur $2k$.

— Si x est de longueur $2k+1$, alors y est de longueur $2k-1$ et par hypothèse de récurrence, y est nécessairement le mot $(ab)^{k-1}a$.
On peut en déduire $x = ab(ab)^{k-1}a = (ab)^k a$ nécessairement.

L'expression régulière $(ab)^*a$ dénote le langage des solutions.
2. On procède par analyse-synthèse.
Analyse :
Soit x une solution de l'équation.
On commence par s'intéresser à la longueur des mots. On a $xx = UxV$ donc $2|x| = |U| + |x| + |V|$ donc $|x| = |UV|$.
Par identification lettre à lettre, on obtient que U préfixe de x et V suffixe de x.
Combiné à la condition sur la longueur, on obtient que nécessairement, $x = UV$.
On a obtenu dans l'analyse l'unicité de la solution si elle existe.
Synthèse :
Vérifions dans quel cas UV est bien solution.
UV est solution lorsque $(UV)(UV) = U(UV)V$, soit $UV = VU$.
C'est la condition nécessaire et suffisante d'existence.

Corrigé

3. Tant que $A_0 = B_0$, on peut simplifier de part et d'autre A_0x et B_0x et décaler les indices.
Ainsi, on se ramène au cas où $A_0 \neq B_0$.
Si $|A_0| = |B_0|$ et $A_0 \neq B_0$, l'ensemble de solution est vide et $A_0 \neq \varepsilon$, et $A_0x = \varepsilon$ est une équation équivalente.
Si $|A_0| < |B_0|$, on se ramène au cas $|A_0| > |B_0|$ en échangeant le rôle des A_i et des B_i (symétrie de la relation $=$).
Dans le cas où $|A_0| > |B_0|$, si B_0 n'est pas préfixe de A_0 , l'ensemble de solution est vide et $A_0 \neq \varepsilon$, et $A_0x = \varepsilon$ est une équation équivalente.
Sinon, B_0 est préfixe de A_0 et on peut simplifier par B_0 de part et d'autre et se ramener au cas $A_0 \neq \varepsilon$ et $B_0 = \varepsilon$.

4. Soit A_0 un mot non vide.
Montrons par récurrence sur $|x|$ que si x est solution d'une équation de type (2) faisant intervenir le mot A_0, alors il existe $k \in \mathbb{N}$ et $0 \leqslant l < |A_0|$ tels que $x = A_0^k A_0[1 : l]$.
Initialisation :
Dans le cas où $|x| < |A_0|$, en identifiant le début $A_0x...$ et le début $xB_1...$ lettre par lettre, on obtient $x = A_0[1 : |x|] = A_0^0 A_0[1 : |x|]$.
En particulier si $x = \varepsilon$, on a bien $x = A_0^0 A_0[1 : 0]$.
Hérédité :
Soit x une solution de longueur $|x| \geqslant |A_0|$, on suppose le résultat vrai pour toute solution de plus petite longueur pour une équation de type (2) avec le même A_0.
En identifiant les débuts, on obtient que x admet A_0 comme préfixe. Donc $x = A_0y$ avec y de longueur $|x| - |A_0|$.
y est solution de $A_0A_0yA_1A_0y...A_r = A_0yB_1A_0y...A_0yB_s$, soit $A_0yA_1A_0y...A_r = yB_1A_0y...A_0yB_s$.
Par hypothèse de récurrence, il existe $k_y \in \mathbb{N}$ et $0 \leqslant l < |A_0|$ tel que $y = A_0^{k_y} A_0[1 : l]$.
Ainsi, $x = A_0 A_0^{k_y} A_0[1 : l] = A_0^{k_y+1} A_0[1 : l]$, donc il existe bien $k = k_y + 1$ et l comme souhaité.

5. On applique le résultat de la question précédente.
Si x est solution alors il existe des entiers n, m tels que $x = (R^k)^n (R^k)[1 : m]$.
On peut simplifier cette écriture en remarquant que $(R^k)[1 : m] = R^i R[1 : j]$ pour i, j vérifiant $i \times |R| + j = m$ et $j < |R|$.
On écrira donc $x = R^p R[1 : j]$ avec p un entier et $0 \leqslant j < |R|$.
On injecte cette solution dans l'équation : $R^k R^p R[1 : j] = R^p R[1 : j] R^k$.
On simplifie par R^p : $R^k R[1 : j] = R[1 : j] R^k$.
On obtient que $R[1 : j]$ est suffixe de R en identifiant la fin donc que $R = R[1 : |R| - j]R[1 : j]$.
Et que $R = R[1 : j]R[1 : |R| - j]$ en identifiant le début.
Donc $RR = R[1 : j]R[1 : |R| - j]R[1 : j]R[1 : |R| - j] = R[1 : j]RR[1 : |R| - j]$.
Comme R est primitif $R[1 : j] = \varepsilon$ ou $R[1 : |R| - j] = \varepsilon$ donc $j = 0$ ou $j = |R|$, mais ce deuxième cas est impossible.
Ainsi, on en déduit $x = R^p R[1 : j] = R^p$ et $x \in \{R\}^*$.
On a montré que l'ensemble des solutions est inclus dans $\{R\}^*$.
Montrons maintenant que, réciproquement, tout mot de $\{R\}^*$ est solution.
Soit $x = R^p \in \{R\}^*$.
On a $R^k R^p = R^{k+p} = R^p R^k$ donc x est bien solution.

6. Montrons le résultat par récurrence forte sur $|P|$.
Initialisation :
Pour P composé d'une lettre, on a bien P primitif et $R = P$ convient.
Hérédité :
Soit P un mot. On suppose le résultat vrai pour tout mot de plus petite longueur strictement.
Si P est primitif, on peut choisir $R = P$.

Sinon, il existe u, v des mots non vides tels que $P^2 = uPv$.
D'après la question **2.**, on a $P = uv = vu$ (P est solution de $xx = uxv$).
Comme u et v sont non vides, alors u et v sont de longueurs strictement plus petites que P et on peut appliquer l'hypothèse de récurrence à u par exemple.
Ainsi, il existe R primitif et k un entier tel que $u = R^k$.
Donc v est solution de $R^k x = xR^k$ donc v est de la forme $R^{k'}$ d'après la question **5.**.
Enfin, $P = R^k R^{k'}$ donc $P = R^{k+k'}$ avec R primitif et $k + k'$ un entier.
7. Soit $x, y \in \Sigma^*$ tels que $xPy = P^k$.
Par identification lettre par lettre, on a $x = P^i P[1 : j]$ avec $i \times |P| + j = |x|$ et $0 \leqslant j < |P|$ et $y = P[j + 1 : |P|]P^{k-2-i}$.
Car $|x| + |y| = (k - 1)|P| = i \times |P| + j + (k - 2 - i) \times |P| + |P| - j$.
Ainsi, $P^k = P^i P[1 : j]PP[j + 1 : |P|]P^{k-2-i}$ et en simplifiant, $P[1 : j]PP[j + 1 : |P|] = PP$.
Or P est primitif donc $j = 0$ nécessairement.
Ainsi, (x, y) nécessairement de la forme (P^i, P^{k-1-i}).
De plus, tous ces couples sont solutions.
8. On peut s'intéresser à la longueur :

$$|A_0| + |x| + \cdots + |A_r| = |x| + |B_1| + |x| + \cdots + |B_s|$$

$$r \times |x| + |A_0| + \cdots + |A_r| = s \times |x| + |B_1| + \cdots + |B_s|$$

$$|x| = \frac{(|B_1| + \cdots + |B_s|) - (|A_0| + \cdots + |A_r|)}{r - s}$$

Ainsi, s'il y a une solution, il n'en existe qu'une de la forme $A_0^k A_0[1 : l]$ de longueur calculée ci-dessus.
Si cet unique mot n'est pas solution, l'ensemble solution est vide.
9. L'équation se ramène à $A_0 x = xB_1$.
Si $|A_0| \neq |B_1|$, il n'y a pas de solution.
Sinon, elles sont de la forme $A_0^k A_0[1 : l]$.
En injectant, on obtient $A_0 A_0^k A_0[1 : l] = A_0^k A_0[1 : l]B_1$, qui se simplifie en $A_0 A_0[1 : l] = A_0[1 : l]B_1$, qui est une condition nécessaire et suffisante d'existence des solutions $A_0^k A_0[1 : l]$ pour tout k et pour une valeur de l fixée.
10. L'équation se ramène à $A_0 x A_1 x = xB_1 xB_2$.
Si $|A_0| + |A_1| \neq |B_1| + |B_2|$, il n'y a pas de solutions.
Sinon, elles sont de la forme $A_0^k A_0[1 : l]$.
En injectant, on obtient $A_0 A_0^k A_0[1 : l]A_1 A_0^k A_0[1 : l] = A_0^k A_0[1 : l]B_1 A_0^k A_0[1 : l]B_2$.
Que l'on peut simplifier en $A_0 A_0[1 : l]A_1 A_0^k A_0[1 : l] = A_0[1 : l]B_1 A_0^k A_0[1 : l]B_2$.
On pourrait aller plus loin, mais il s'agit d'une condition nécessaire et suffisante d'existence pour un couple de valeurs (l, k).

Corrigé

Chapitre 2
Automates finis

UN SCIENTIFIQUE

Edward Moore (1925-2003)

Délaissant la chimie, Edward Moore obtient un doctorat en mathématiques en 1950. Il travaille alors avec Claude Shannon sur la calculabilité, théorie qui recherche les limites de ce qui peut être obtenu par le biais d'un algorithme. Il étudie les automates finis, ce qui explique que plusieurs notions portent son nom dans ce domaine. On lui doit de nombreux algorithmes, en particulier celui de minimisation d'un automate fini. John Conway s'inspirera de ses travaux pour concevoir le jeu de la vie.

Un peu d'histoire

La notion d'automate fini peut modéliser des situations de changement d'états dans de nombreuses disciplines et trouve de multiples applications concrètes comme le fonctionnement d'un ascenseur, d'un distributeur de monnaie ou d'un digicode. Cette formalisation a pris de l'importance dans le cadre de l'informatique théorique. Cependant, au tournant du neuvième siècle, Alcuin, le conseiller de Charlemagne avait déjà posé un problème qui est un bel exemple pouvant être modélisé par cette méthode.

« Un berger, revenant du marché avec un loup, une chèvre et un chou, doit traverser une rivière pour rentrer chez lui. Il dispose d'une barque juste assez grande pour le transporter avec un seul de ses trois achats. Ni la chèvre et le chou, ni le loup et la chèvre, ne peuvent se trouver seuls sur une rive, sans quoi l'un des deux se ferait dévorer. Qui peut dire comment le berger peut traverser la rivière sans dégâts ? »

OBJECTIFS

- Construire, analyser, exploiter et manipuler les automates finis :
  - Comprendre le fonctionnement des automates finis déterministes et non déterministes et savoir passer de l'un à l'autre via un procédé de déterminisation
  - Construire un automate fini à partir d'une expression régulière
  - Retrouver une expression régulière à partir d'un automate
  - Démontrer les propriétés de stabilité des langages réguliers en s'appuyant sur les automates
  - Raisonner sur la structure d'automate fini
  - Montrer qu'un langage n'est pas régulier en utilisant notamment le lemme de l'étoile

Cours

Automates finis déterministes

Définitions

Définitions : *Un* ***automate fini déterministe*** *est défini par un quintuplet* $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$*, où :*

- Σ *est un* ***alphabet fini*** *;*
- Q *est un ensemble fini dont les éléments sont appelés des* ***etats@états*** *;*
- $q_0 \in Q$ *est un état particulier dit* ***initial*** *;*
- $F \subset Q$ *est un ensemble d'états dits* ***finaux*** *(ou acceptants) ;*
- δ *est une fonction d'une partie de* $Q \times \Sigma$ *vers* Q *appelée* ***fonction de transition****.*

Lorsque δ *est définie sur* $Q \times \Sigma$ *tout entier, l'automate est dit* ***complet*** *.*

On représente les automates par des graphes orientés dont les sommets sont les états. Pour tout $(q_i, a, q_j) \in Q \times \Sigma \times Q$ tel que $\delta(q_i, a) = q_j$, le graphe possède un arc (q_i, q_j) d'étiquette a. Il s'agit d'une transition de q_i à q_j d'étiquette a. On indique l'état initial par une flèche entrante et les états finaux par un double cercle. D'autres conventions existent, nous retenons celle-ci dans ce livre.

Exemple : Voici un automate complet, la représentation de la fonction de transition sous forme de tableau, ainsi que sa représentation sous la forme d'un graphe.

$$\Sigma = \{a, b\} \quad Q = \{q_0, q_1, q_2\} \quad F = \{q_0, q_2\}$$

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

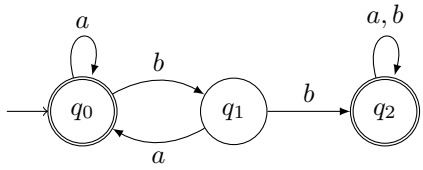


FIGURE 2.1 – Exemple d'automate fini déterministe.

Remarque : Dans la littérature, on peut représenter les transitions par un sous-ensemble Δ de $Q \times \Sigma \times Q$. Dans ce cas, $(q, a, p) \in \Delta$ a le même sens que $\delta(q, a) = p$. On parle alors d'***ensemble de transitions***.
Le caractère **déterministe** est garanti par la propriété de Δ :

$$\forall (q, a, p, p') \in Q \times \Sigma \times Q \times Q, \quad (q, a, p) \in \Delta \quad \text{et} \quad (q, a, p') \in \Delta \quad \text{implique} \quad p = p'$$

Cela traduit le fait que δ est une fonction : une image au plus pour un couple (q, a). Autrement dit, l'automate a au plus une transition sortant d'un état étiquetée par une lettre donnée.

Définitions : *Un **chemin** dans un automate est une suite finie de transitions de la forme :*

$$q_{i_0} \xrightarrow{a_1} q_{i_1} \xrightarrow{a_2} \dots \xrightarrow{a_n} q_{i_n}$$

L'étiquette du chemin est le mot $a_1...a_n$ formé par la suite de lettres étiquetant les transitions.
*Le chemin est **acceptant** lorsque l'état de départ est l'état initial ($q_{i_0} = q_0$) et l'état d'arrivée est un état final ($q_{i_n} \in F$).*
*Un mot sur l'alphabet Σ est **reconnu** par l'automate lorsqu'il est l'étiquette d'un chemin acceptant.*
*$\mathcal{L}(\mathcal{A})$, le **langage reconnu** par l'automate $\mathcal{A}$, est l'ensemble des mots reconnus par $\mathcal{A}$.*
*On parle aussi de **calcul** sur l'automate : $q_{i_0} \xrightarrow{a_1} q_{i_1} \xrightarrow{a_2} \dots \xrightarrow{a_n} q_{i_n}$ calcule le mot $a_1...a_n$.*

Exemple : L'ensemble des mots reconnus par l'automate de l'exemple précédent (figure 2.1) est dénoté par l'expression régulière $(a|ba)^*(\varepsilon|(bb(a|b)^*))$. Il s'agit de l'ensemble des mots vides ou contenant le facteur bb ou terminant par un a.

Remarque : On dit que ces automates sont déterministes car pour un mot donné, il n'existe qu'un chemin possible depuis l'état initial. L'état d'arrivée est alors unique et permet de savoir si le mot est reconnu ou non.

Remarque : Soit $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ un automate.
On peut définir récursivement la ***fonction de transition étendue*** $\delta^* : Q \times \Sigma^* \to Q \cup \{\perp\}$ par :

- $\forall q \in Q, \quad \delta^*(q, \varepsilon) = q$;
- $\forall q \in Q, \forall u \in \Sigma^*, \forall a \in \Sigma, \quad \delta^*(q, au) = \delta^*(\delta(q, a), u)$ si $\delta(q, a)$ est défini ;
- $\forall q \in Q, \forall u \in \Sigma^*, \quad \delta^*(q, u) = \perp$ s'il n'existe pas de chemin d'étiquette u d'origine q dans $\mathcal{A}$.

La fonction de transition étendue n'est pas mentionnée dans le programme de MPI, il s'agit donc de l'introduire si elle n'est pas déjà introduite aux concours.

Un automate peut être abordé comme une machine qui lit un mot lettre par lettre. Un mot est accepté si le chemin suivi est acceptant et refusé si ce chemin n'est pas acceptant ou inexistant.

■ Émondage d'un automate fini

Soit $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ un automate fini déterministe.

Définitions : *Un état q est **accessible** lorsqu'il existe un chemin depuis l'état initial menant à cet état q. Un état q est **co-accessible** lorsqu'il existe un chemin depuis q menant à un état final.*

Exemple : Considérons l'automate de la figure 2.2 ci-contre, qui n'est pas complet.
L'état q_4 n'est pas accessible, mais il est co-accessible car final.
L'état q_3 n'est pas co-accessible, mais il est accessible, par un chemin d'étiquette a par exemple.

Définition : *Les états à la fois accessibles et co-accessibles sont appelés les états **utiles**.*

Proposition 2.1.— Enlever les états non accessibles ou non co-accessibles d'un automate ne modifie pas le langage reconnu.

Démonstration
Les chemins acceptants restent les mêmes car ils n'exploitent que des états utiles. ▲

Définition : *On appelle **automate émondé** l'automate obtenu en supprimant les états qui ne sont pas des états utiles.*

Remarque : L'automate émondé n'est pas nécessairement complet.

$$\Sigma = \{a, b\} \qquad Q = \{q_0, q_1, q_2, q_3, q_4\} \qquad F = \{q_2, q_4\}$$

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

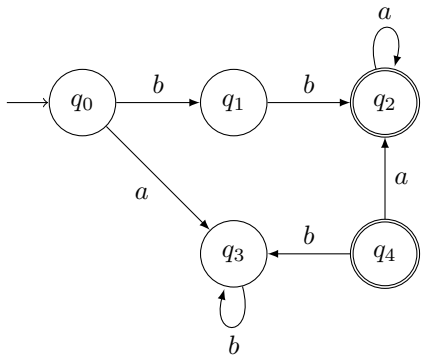


FIGURE 2.2 – États accessibles et co-accessibles dans un automate.

Exemple : L'automate émondé de l'automate de l'exemple précédent est le suivant : on a enlevé les états q_3 et q_4 et les transitions associées.

$$\Sigma = \{a, b\} \qquad Q = \{q_0, q_1, q_2\} \qquad F = \{q_2\}$$

δ	a	b
q_0	-	q_1
q_1	-	q_2
q_2	q_2	-

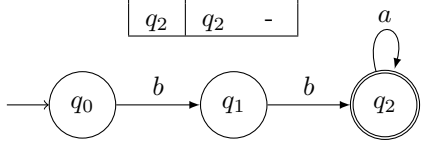


FIGURE 2.3 – Automate émondé de l'automate de la figure 2.2

Remarque : L'état q_3 que l'on a supprimé est ce que l'on peut appeler un ***état puits*** : il n'est pas final et on ne peut pas en sortir.

Définition : *Pour un couple $(q, a) \in Q \times \Sigma$, si $\delta(q, a)$ n'est pas défini alors (q, a) est un* ***blocage*** *dans l'automate.*

⚠ **Remarque :** Pour rendre un automate complet, autrement dit compléter un automate, il suffit d'ajouter un état puits et pour chaque blocage ajouter une transition vers l'état puits.

■ Automates finis non déterministes (avec ou sans ε-transitions)

Définition : *Un* ***automate fini non déterministe*** *(sans ε-transitions) est défini par un quintuplé $\mathcal{A} = (\Sigma, Q, I, F, \delta)$, où :*

- *Σ est un* ***alphabet fini*** *;*
- *Q est un ensemble fini dont les éléments sont appelés des* ***états*** *;*
- *$I \subset Q$ est un ensemble d'états* ***initiaux*** *;*
- *$F \subset Q$ est un ensemble d'états* ***finaux*** *;*
- *δ est une fonction de $Q \times \Sigma$ dans $\mathcal{P}(Q)$ appelée* ***fonction de transition****.*

On a une transition de $q \in Q$ à $p \in Q$ d'étiquette $a \in \Sigma$ lorsque $p \in \delta(q, a)$. On note alors $q \xrightarrow{a} p$. Les notions de chemins et de langages acceptés sont les mêmes qu'avec les automates déterministes.

⚠ **Remarques :** Il y a deux différences majeures entre les automates déterministes et les automates non déterministes :

- dans un automate déterministe, on a unicité de l'état initial, alors que pour un automate non déterministe, on peut avoir plusieurs états initiaux ;
- la fonction de transition est à valeur dans Q pour un automate déterministe et dans $\mathcal{P}(Q)$ pour un automate non déterministe.

Dans un automate non déterministe, il est possible qu'il existe plusieurs chemins avec la même étiquette depuis un état initial.
Dans la littérature, les transitions peuvent être représentées par un sous-ensemble Δ de $Q \times \Sigma \times Q$, appelé ***ensemble de transitions***. Dans ce cas, $(q, a, p) \in \Delta$ a le même sens que $p \in \delta(q, a)$.

Définition : *Un automate fini non déterministe est dit* ***complet*** *lorsque $\forall (q, a) \in Q \times \Sigma$, $\delta(q, a) \neq \emptyset$.*

Exemple : Voici un exemple d'automate non déterministe avec une représentation de la fonction de transition par un tableau et une représentation graphique de l'automate.

$$\Sigma = \{a, b\} \quad Q = \{q_0, q_1, q_2\} \quad I = \{q_0, q_2\} \quad F = \{q_2\}$$

δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

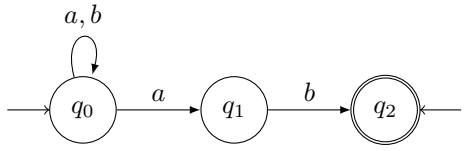


FIGURE 2.4 – Exemple d'automate non déterministe.

Le mot ab peut être l'étiquette de deux chemins commençant en un état initial :

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \quad \text{et} \quad q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2.$$

En particulier, le premier n'est pas acceptant, mais le deuxième est acceptant. Donc ab est reconnu par l'automate.

Définition : *Un **automate fini non déterministe avec ε-transitions** est défini par un quintuplet $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ de la même façon qu'un automate fini non déterministe mais avec δ une fonction de $Q \times (\Sigma \cup \{\varepsilon\})$ dans $\mathcal{P}(Q)$, encore appelée fonction de transition.*
*Une **transition spontanée** $q_1 \xrightarrow{\varepsilon} q_2$, aussi appelée **$\varepsilon$-transition**, est le passage d'un état $q_1 \in Q$ à un état $q_2 \in Q$ sans lecture d'une lettre et correspond à la situation où $q_2 \in \delta(q_1, \varepsilon)$.*

Exemple : L'automate suivant est non déterministe avec ε-transition et reconnaît le langage $\{ab, b\}$.

$$\Sigma = \{a, b\} \qquad Q = \{q_0, q_1, q_2\} \qquad I = \{q_0\} \qquad F = \{q_2\}$$

δ	a	b	ε
q_0	$\{q_1\}$	-	$\{q_1\}$
q_1	-	$\{q_2\}$	-
q_2	-	-	-

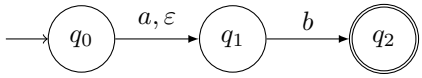


FIGURE 2.5 – Automate non déterministe avec ε-transition, reconnaissant le langage $\{ab, b\}$.

⚠ **Remarque :** Il faut bien faire la différence entre les automates finis avec ou sans ε-transitions. En effet, les modèles sont bien différents. Dans certains exercices et sujets de concours, la définition imposée des automates n'autorise pas les ε-transitions : il ne faut pas en utiliser dans ce cas.

■ Déterminisation

■ Déterminisation des automates sans transitions spontanées

Pour créer un automate déterministe $\mathcal{A}_D$ à partir d'un automate non déterministe (sans transition spontanée dans un premier temps) $\mathcal{A}_{ND} = (\Sigma, Q, I, F, \delta)$, on pose :

- $Q_D = \mathcal{P}(Q)$ les sous-ensembles d'états de l'automate initial ;
- $q_{0,D} = I \in Q_D$ le singleton contenant l'ensemble des états initiaux de l'automate initial ;
- $F_D = \{P \in Q_D, P \cap F \neq \emptyset\}$ l'ensemble des parties de Q contenant au moins un état final ;
- δ_D la fonction de transition définie de $Q_D \times \Sigma$ dans Q_D par :

$$\delta_D(P, a) = \{q' \in Q \mid \exists q \in P, q' \in \delta(q, a)\} = \bigcup_{q \in P} \delta(q, a)$$

$\mathcal{A}_D = (\Sigma, Q_D, q_{0,D}, F_D, \delta_D)$ est appelé ***automate des parties*** car ses états sont les parties de Q.

Remarque : L'automate des parties $\mathcal{A}_D$ est déterministe et complet par construction.

Proposition 2.2.— S'il existe un chemin de $q \in Q$ à $q' \in Q$ d'étiquette $u \in \Sigma^*$ dans $\mathcal{A}_{ND}$ alors il existe $P, P' \in Q_D$ tels que $q \in P$ et $q' \in P'$ et un chemin d'étiquette u de P à P' dans $\mathcal{A}_D$.

Démonstration
On note $u = a_1 ... a_n$ où les a_i sont les lettres de u et $q = q_{i_0} \xrightarrow{a_1} q_{i_1} \xrightarrow{a_2} \dots \xrightarrow{a_n} q_{i_n} = q'$ un chemin dans $\mathcal{A}_{ND}$. Montrons le résultat par récurrence sur la longueur n du chemin.
Initialisation :
Si $n = 0$, nécessairement $q = q'$ et on a un chemin vide d'étiquette ε de P à P pour tout $P \in Q_D$ tel que $q \in P$, en particulier pour $P = \{q\}$.

Hérédité :
On suppose le résultat vrai pour $n-1 \geqslant 0$.
Par hypothèse de récurrence, il existe un chemin de $P \in Q_D$ à $P'' \in Q_D$ d'étiquette $a_1...a_{n-1}$ tel que $q_{i_0} \in P$ et $q_{i_{n-1}} \in P''$. On complète ce chemin par une transition adéquate :
$P'' \xrightarrow{a_n} \delta(P'', a_n) = \{p' \in Q \mid \exists p'' \in P'', p' \in \delta(p'', a_n)\} = P'$ et on a bien $q' \in P'$ car $q_{i_{n-1}} \in P''$ et $q_{i_n} \in \delta(q_{i_{n-1}}, a_n)$. Ainsi, on obtient le résultat attendu.
Par principe de récurrence, le résultat est vrai pour tout n. ▲

Proposition 2.3.— Soit P, P' deux sous-ensembles de Q non vides. S'il existe un chemin dans $\mathcal{A}_D$ partant de P et menant à P' par un mot u alors pour tout $q' \in P'$, il existe un état $q \in P$ tel qu'il existe un chemin entre q et q' dans $\mathcal{A}_{ND}$.

Démonstration
On note $u = a_1...a_n$, les a_i étant les lettres de u, et $P = P_{i_0} \xrightarrow{a_1} P_{i_1} \xrightarrow{a_2} \dots \xrightarrow{a_n} P_{i_n} = P'$ un chemin dans $\mathcal{A}_D$. Montrons le résultat par récurrence sur n la longueur du chemin.
Initialisation :
Si $n = 0$, nécessairement $P = P'$ et $u = \varepsilon$.
Pour tout $q \in P$, on choisit $q' = q \in P'$, et on a bien un chemin d'étiquette ε dans $\mathcal{A}_{ND}$ de q à q'.
Hérédité :
On suppose le résultat vrai pour $n-1 \geqslant 0$.
Par hypothèse de récurrence, pour tout $q'' \in P_{i_{n-1}}$, il existe $q \in P_{i_0}$ tel qu'il existe un chemin dans $\mathcal{A}_{ND}$ d'étiquette $a_1...a_{n-1}$ de q à q''.
De plus $P_{i_n} = \delta(P_{i_{n-1}}, a_n) = \{q' \in Q_D \mid \exists q'' \in P_{i_{n-1}}, q' \in \delta(q'', a_n)\}$ par définition et $P_{i_n} \neq \emptyset$ par hypothèse. Soit donc $q' \in P_{i_n}$.
Par définition, il existe $q'' \in P_{i_{n-1}}$ tel que $q' \in \delta(q'', a_n)$. Ainsi, il existe une transition $q'' \xrightarrow{a_n} q'$ que l'on utilise pour compléter un chemin d'un certain $q \in P_{i_0}$ à q'' qui existe par hypothèse de récurrence.
Ainsi, on obtient le résultat attendu pour tout $q' \in P_{i_n}$.
Par principe de récurrence, le résultat est vrai pour tout n. ▲

Proposition 2.4.— L'automate $\mathcal{A}_{ND}$ et l'automate $\mathcal{A}_D$ reconnaissent le même langage.

Démonstration
Les deux précédentes propriétés permettent de montrer ce résultat par double inclusion. L'état $\emptyset$ est un état puits et n'apparaît pas dans les chemins acceptants. ▲

Pour déterminiser un automate, on peut ainsi construire l'automate des parties.

Exemple : Construisons l'automate des parties pour l'automate exemple suivant.

$$\Sigma = \{a, b\} \qquad Q = \{q_0, q_1, q_2\} \qquad I = \{q_0, q_2\} \qquad F = \{q_2\}$$

δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	-	$\{q_2\}$
q_2	-	-

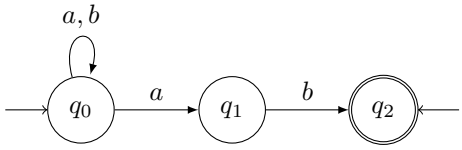


FIGURE 2.6 – Automate reconnaissant le langage des mots sur $\{a, b\}$ finissant par ab.

Pour cela, on construit d'abord le tableau correspondant avec une ligne par sous-ensemble d'états et une colonne par lettre de l'alphabet pour déterminer la fonction de transition.

δ_d	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

On repère ensuite l'état initial, qui est l'ensemble des états initiaux, ici $\{q_0, q_2\}$, et les états finaux qui sont tous les états contenant au moins un état final de l'automate original, ici ceux contenant q_2.

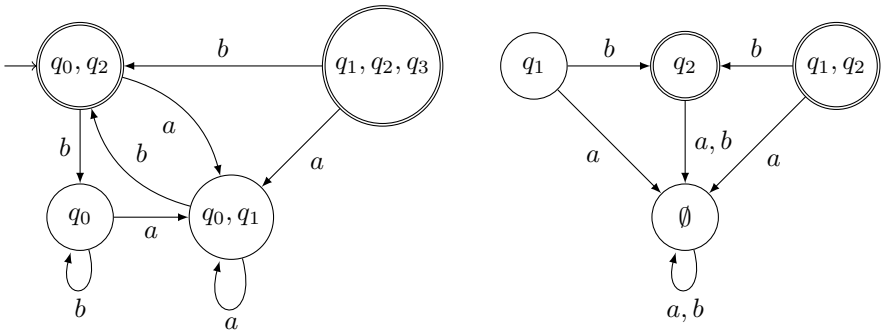


FIGURE 2.7 – Automate des parties de l'automate de la figure 2.6.

On remarque que l'automate est représenté par deux composantes connexes dont l'une n'est pas accessible depuis l'état initial et est donc émondable. L'état $\{q_1, q_2, q_3\}$ est également émondable.

⚠ **Remarque :** On peut se limiter aux états accessibles en partant de I et en déterminant les transitions de proche en proche sur les états accessibles. Dans l'exemple précédent, cela nous limite aux états $\{q_0, q_2\}$,$\{q_0, q_1\}$ et $\{q_0\}$. Sinon, il y a beaucoup à émonder.
En pratique, il n'est pas nécessaire de construire l'automate des parties en entier, on construit le tableau à partir d'une ligne pour I puis on ajoute des lignes au fur et à mesure que l'on découvre de **nouveaux états accessibles** dans l'automate des parties.

■ Élimination des transitions spontanées

Dans le cas d'un automate non déterministe avec ε-transitions, il faut préalablement éliminer les transitions spontanées.

Définitions : *L'**ε-fermeture** d'un état q est l'ensemble des états obtenus en suivant les ε-transitions partant de q. Autrement dit, les états accessibles depuis q sans lire de lettre.*
En particulier, on peut définir rigoureusement l'ε-fermeture $F_\varepsilon(q)$ de q par induction :

- $q \in F_\varepsilon(q)$;
- *si* $p \in F_\varepsilon(q)$ *et* $r \in \delta(p, \varepsilon)$ *alors* $r \in F_\varepsilon(q)$.

*On dit qu'un ensemble est **clos par ε-fermeture** s'il contient l'ε-fermeture de tous ses éléments.*