

Automates à états finis et langages réguliers

**Rappels des notions essentielles
et plus de 170 exercices corrigés**

Yliès Falcone

Maître de conférences à
l'Université Grenoble Alpes

Membre du Laboratoire d'Informatique de Grenoble
et d'Inria Grenoble Rhône-Alpes

Jean-Claude Fernandez

Ancien Professeur à l'Université Grenoble Alpes
Ancien membre du laboratoire Verimag

DUNOD

Illustration de couverture :
© Anita Ponne – Adobe Stock

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocopillage.

Le Code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements

d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée. Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).



© Dunod, 2020
11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-080846-5

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2^o et 3^o al), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

Table des matières

Avant-Propos	9
Introduction	13
1 Rappels et notations	19
1.1 Notions de logique, assertions	19
1.2 Quantificateurs	21
1.3 Notions ensemblistes	22
1.4 Entiers naturels	25
1.5 Ensembles définis inductivement	26
1.6 Raisonnements, techniques de preuve	26
2 Notions préliminaires	29
2.1 Résumé intuitif du chapitre	29
2.2 Les notions essentielles	29
2.3 Exercices	32
2.3.1 Mots et concaténation	32
2.3.2 Langages	33
2.3.3 Concaténation de langages	34
2.4 Indications pour résoudre les exercices	35
2.5 Solutions des exercices	37
2.5.1 Mots et concaténation	37
2.5.2 Langages	41
2.5.3 Concaténation de langages	42
3 Automates déterministes	45
3.1 Résumé intuitif du chapitre	45
3.2 Les notions essentielles	46
3.2.1 Définition et langage reconnu	46
3.2.2 Accessibilité et coaccessibilité	48
3.3 Exercices	49

3.3.1	Automate et langage reconnu	49
3.3.2	Automate reconnaissant un langage	50
3.3.3	Fonction de transition d'un automate	51
3.3.4	Langage à états ou non	53
3.3.5	Accessibilité et coaccessibilité	53
3.3.6	Automates et langages particuliers	53
3.4	Indications pour résoudre les exercices	54
3.5	Solutions des exercices	55
3.5.1	Automate et langage reconnu	55
3.5.2	Automate reconnaissant un langage	59
3.5.3	Fonction de transition d'un automate	66
3.5.4	Accessibilité et coaccessibilité	69
3.5.5	Automates et langages particuliers	70
4	Opérations sur les automates déterministes	71
4.1	Résumé intuitif du chapitre	71
4.2	Les notions essentielles	72
4.3	Exercices	74
4.3.1	Automate complet / complété	74
4.3.2	Automate complémentaire	74
4.3.3	Automate produit	75
4.4	Indications pour résoudre les exercices	76
4.5	Solutions des exercices	77
4.5.1	Automate complété	77
4.5.2	Automate complémentaire	81
4.5.3	Automate produit	85
5	Algorithmes sur les automates déterministes	89
5.1	Résumé intuitif du chapitre	89
5.2	Les notions essentielles	90
5.2.1	Successseurs et prédecesseurs d'un état	90
5.2.2	Parcours	90
5.2.3	Notions d'accessibilité et de coaccessibilité	93
5.2.4	Détection de cycles	97
5.2.5	Quelques problèmes de décision	98
5.3	Exercices	101
5.3.1	Complétion, complémentation et produit	101
5.3.2	Émonder un automate	102
5.3.3	Inclusion et égalité des langages reconnus par des automates	102
5.3.4	Autour de la notion de préfixe	103
5.4	Indications pour résoudre les exercices	103
5.4.1	Compléter un automate	103
5.4.2	Émonder un automate	104
5.4.3	Inclusion et égalité des langages reconnus par des automates	105
5.4.4	Autour de la notion de préfixe	105
5.5	Solutions des exercices	106

5.5.1	Compléter un automate	106
5.5.2	Émonder un automate	110
5.5.3	Inclusion et égalité des langages reconnus par des automates	112
5.5.4	Autour de la notion de préfixe	113
6	Minimisation d'automates déterministes	119
6.1	Résumé intuitif du chapitre	119
6.2	Les notions essentielles	120
6.2.1	Relation de distinguabilité entre états	121
6.2.2	Relation d'équivalence entre états d'un AFED	123
6.2.3	Minimisation	125
6.2.4	Tester l'équivalence entre deux AFED	125
6.3	Exercices	126
6.3.1	Équivalence et distinguabilité entre états	126
6.3.2	Minimisation	127
6.3.3	Équivalence et distinguabilité entre AFED	129
6.4	Indications pour résoudre les exercices	129
6.5	Solutions des exercices	130
6.5.1	Équivalence et distinguabilité entre états	130
6.5.2	Minimisation	130
6.5.3	Équivalence et distinguabilité	135
7	Automates non déterministes	143
7.1	Résumé intuitif du chapitre	143
7.2	Les notions essentielles	144
7.2.1	Définition et langage reconnu	144
7.2.2	Déterminisation des automates non déterministes	145
7.3	Exercices	146
7.3.1	Langage et AFEND	146
7.3.2	Déterminiser un AFEND	148
7.3.3	Équivalence entre AFEND	149
7.3.4	Complexité	150
7.3.5	Algorithmes pour les AFEND	151
7.4	Indications pour résoudre les exercices	151
7.5	Solutions des exercices	153
7.5.1	Langage et AFEND	153
7.5.2	Déterminiser un AFEND	158
7.5.3	Déterminer l'équivalence entre deux AFEND	161
7.5.4	Complexité	162
7.5.5	Algorithmes pour les AFEND	166
8	Automates non déterministes avec ϵ-transitions	167
8.1	Résumé intuitif du chapitre	167
8.2	Les notions essentielles	168
8.2.1	Définition et langage reconnu	168

8.2.2	Élimination des ϵ -transitions	168
8.2.3	Retour sur la fermeture de la classe des langages à états	170
8.3	Exercices	172
8.3.1	Trouver un automate	173
8.3.2	Fermeture de Kleene	175
8.3.3	Élimination des ϵ -transitions et déterminisation	175
8.3.4	Composition et transformation d' ϵ -AFEND	177
8.3.5	Algorithmes sur les ϵ -AFEND	179
8.4	Indications pour résoudre les exercices	179
8.5	Solutions des exercices	182
8.5.1	Trouver un automate	182
8.5.2	Fermeture de Kleene	186
8.5.3	Élimination des ϵ -transitions et déterminisation	189
8.5.4	Composition et transformation d' ϵ -AFEND	193
8.5.5	Algorithmes sur les ϵ -AFEND	201
9	Expressions régulières	205
9.1	Résumé intuitif du chapitre	205
9.2	Les notions essentielles	206
9.2.1	Syntaxe des expressions régulières	206
9.2.2	Sémantique des expressions régulières	206
9.2.3	Conventions de notation et précédence des opérateurs	207
9.2.4	Quelques propriétés : équivalence et simplification	207
9.2.5	Quelques problèmes de décision : équivalence et inclusion de langages dénotés	207
9.3	Exercices	208
9.4	Indications pour résoudre les exercices	210
9.5	Solutions des exercices	211
10	Théorème de Kleene	219
10.1	Résumé intuitif du chapitre	219
10.2	Les notions essentielles	220
10.2.1	Théorème de Kleene	220
10.2.2	Des automates vers les expressions régulières	220
10.2.3	Des expressions régulières vers les automates	224
10.3	Exercices	228
10.3.1	Calcul d'expressions régulières à partir d'automates	229
10.3.2	Calcul d'automates à partir d'expressions régulières	231
10.4	Indications pour résoudre les exercices	232
10.5	Solutions des exercices	233
10.5.1	Calcul d'expressions régulières à partir d'automates	233
10.5.2	Calcul d'automates à partir d'expressions régulières	241

11 Grammaires	247
11.1 Résumé intuitif du chapitre	247
11.2 Les notions essentielles	247
11.2.1 Définition des grammaires	247
11.2.2 Dérivations en une et plusieurs étapes	248
11.2.3 Langage généré par une grammaire	248
11.2.4 Classification de Chomsky des grammaires	248
11.3 Exercices	249
11.3.1 Générer des mots avec les grammaires	249
11.3.2 Langages et grammaires	250
11.4 Indications pour résoudre les exercices	251
11.5 Solutions des exercices	252
11.5.1 Générer des mots avec les grammaires	252
11.5.2 Langages et grammaires	252
12 Grammaires régulières	257
12.1 Résumé intuitif du chapitre	257
12.2 Les notions essentielles	257
12.2.1 Grammaires et automates	257
12.2.2 Grammaires et expressions régulières	260
12.3 Exercices	261
12.3.1 Des grammaires vers les automates	261
12.3.2 Des automates vers les grammaires	261
12.3.3 Des expressions régulières vers les automates et les grammaires	262
12.3.4 Grammaires non régulières	262
12.3.5 Composition de grammaires régulières	263
12.4 Solutions des exercices	263
12.4.1 Des grammaires vers les automates	263
12.4.2 Des automates vers les grammaires	264
12.4.3 Des expressions régulières vers les automates et les grammaires	269
12.4.4 Grammaires non régulières	269
12.4.5 Composition de grammaires régulières	271
13 Propriété de l’itération	273
13.1 Résumé intuitif du chapitre	273
13.2 Les notions essentielles	273
13.2.1 Lemme de l’itération	273
13.2.2 Constante d’itération	274
13.3 Exercices	275
13.3.1 Lemme de l’itération	275
13.3.2 Constante d’itération	275
13.4 Indications pour résoudre les exercices	275
13.5 Solutions des exercices	276
13.5.1 Lemme de l’itération	276
13.5.2 Constante d’itération	277

14 Démontrer la non-régularité	281
14.1 Résumé intuitif du chapitre	281
14.2 Les notions essentielles	281
14.2.1 Utilisation du lemme de l'itération	281
14.2.2 Utilisation des propriétés de fermeture	282
14.2.3 Une condition suffisante pour la non-régularité	283
14.3 Exercices	283
14.3.1 Lemme de l'itération	283
14.3.2 Fermeture des langages réguliers	284
14.3.3 Une condition suffisante pour la non-régularité	286
14.4 Indications pour résoudre les exercices	286
14.5 Solutions des exercices	287
14.5.1 Lemme de l'itération	287
14.5.2 Fermeture des langages réguliers	291
14.5.3 Une condition suffisante pour la non-régularité	294
A Modélisation et résolution de problèmes avec les automates	295
A.1 Exercices	295
A.2 Solutions des exercices	299
Bibliographie	311
Index	313

Avant-Propos

À qui s'adresse ce livre ?

Cet ouvrage s'adresse aux étudiants de premier cycle universitaire suivant un cursus incluant l'informatique (étudiants dans un parcours d'informatique intégral, de mathématiques appliquées ou de mathématiques). Il vise aussi bien les étudiants des IUT, de licences et de classes préparatoires aux grandes écoles. Peu de pré-requis sont nécessaires et correspondent à ceux d'un cours de la théorie des ensembles telle que enseignée en première année de l'enseignement supérieur. Les notions de cours nécessaires sont par ailleurs rappelées dans le premier chapitre.

Langages, automates et informatique

La théorie des langages et des automates revêt un aspect particulier dans l'apprentissage de l'informatique. C'est un sujet qui appartient à la branche fondamentale de l'informatique, tant les concepts abordés se retrouvent dans de nombreuses disciplines de l'informatique aussi bien théoriques qu'appliquées (par exemple la conception des processeurs, la compilation de programmes informatiques, la traduction automatique des langues naturelles, l'intelligence artificielle, la bio-informatique, la vérification de programmes embarqués critiques, la cybersécurité, etc.). Nous développons ce point dans l'introduction en donnant des exemples d'utilisation de cette théorie. La théorie des langages et automates est un enseignement incontournable dans tout cursus d'informatique, tant en France qu'à l'étranger.

Philosophie du livre

L'objectif de cet ouvrage est de conduire le lecteur vers la maîtrise des concepts et techniques de langage à états, ou langage rationnel. Il est orienté vers la pratique d'exercices. Il se veut être un outil accompagnant le travail des étudiants à travers à la fois des exercices simples d'application pour l'appropriation des notions et des exercices plus avancés pour la maîtrise des concepts. Chaque chapitre comporte un rappel des notions essentielles du cours

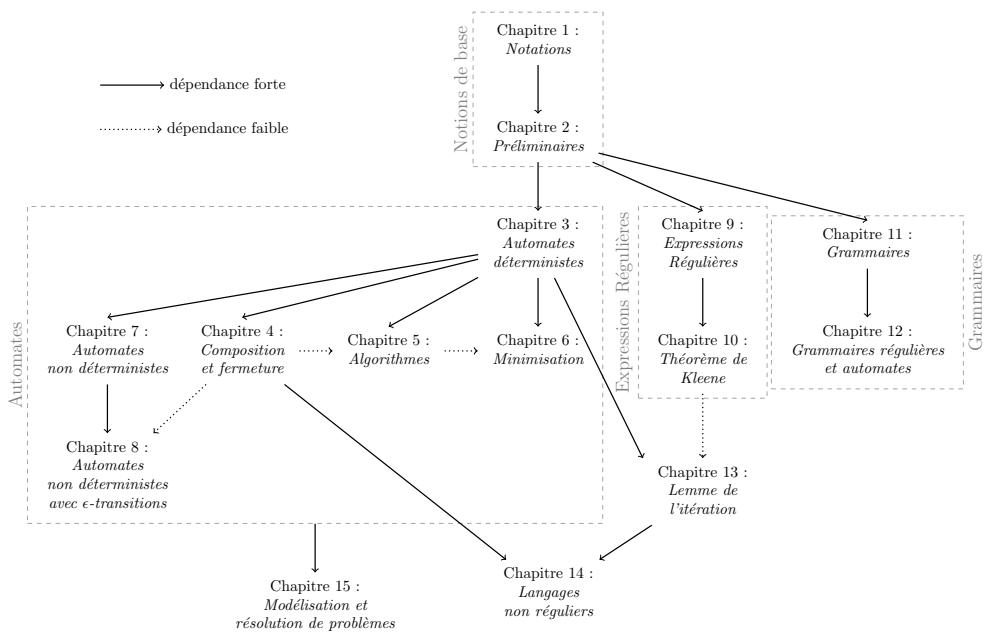


Figure 1 – Plan par chapitre et dépendance entre chapitres.

(nécessaire et suffisant pour la résolution des exercices), les énoncés des exercices et pour chaque exercice, une solution complète. Pour les exercices difficiles, nous fournissons des indications, permettant de débloquer la recherche de la solution sans consulter la solution. Les auteurs sont convaincus des bienfaits de la recherche de solutions, qui développe les compétences et l'apprentissage (et de la relative inutilité de consulter immédiatement une solution sans rechercher la solution au préalable). Les rappels de cours permettent de fixer les notations et donnent les outils pour résoudre les exercices. Les rappels de cours ne comportent pas de démonstrations des théorèmes abordés, mais certaines démonstrations sont traitées dans les exercices. Dans chaque section d'exercices, pour chaque notion, nous proposons des exercices de difficulté croissante, commençant toujours pas des exercices d'application directe du cours pour vérifier la compréhension des concepts. Les solutions d'exercices sont détaillées, permettant le travail autonome du lecteur.

Vue d'ensemble du livre

Vue d'ensemble. Après avoir rappelé les notions indispensables de mathématiques pour pouvoir résoudre les exercices, un premier chapitre est consacré à la notion de langage. Puis nous introduisons les notions d'automates déterministes, non déterministes, non déterministes avec epsilon transitions. Notons que ces trois variations d'automate ont la même puissance d'expression. Un lien est ensuite fait avec la notion d'expressions régulières et de grammaires. Chaque chapitre est structuré de la manière suivante : une partie cours, une partie exercice et une partie solution des exercices.

Chapitre par chapitre. Nous donnons une brève description de chaque chapitre.

Dans le chapitre 1, nous rappelons les notions de mathématiques nécessaires et suffisantes : notions de logique, d'ensembles et de techniques de démonstration. Il n'y a pas d'exercice associé à ce chapitre.

Dans le chapitre 2, nous introduisons les notions d'alphabet, qui est un ensemble fini de symboles, de mots et de langage. Un mot est une composition de symboles appartenant à un alphabet, et un langage un ensemble de mots. Un alphabet est un ensemble fini de symboles et la composition de symboles pour former un mot s'appelle concaténation.

Dans le chapitre 3, nous définissons la notion d'automate déterministe où les règles de transition sont définies par une fonction qui, étant donné un état et une action, détermine l'état suivant. On définit le langage associé à un automate : une configuration est un couple (état, mot) et un mot est accepté si, partant d'une configuration initiale (état initial, mot), on atteint une configuration terminale formée d'un état accepteur et du mot vide.

Dans le chapitre 4, nous définissons les opérations sur les automates permettant de calculer l'intersection, l'union ou le complément de langages associés à un automate.

Dans le chapitre 5, nous présentons les algorithmes sur les automates en lien avec les opérations présentées au chapitre précédent.

Dans le chapitre 6, partant de la notion d'équivalence (ou de distinguabilité) entre états, nous présentons la minimisation des automates, c'est-à-dire que, pour un automate donné, nous pouvons obtenir un automate avec un nombre minimal d'états qui reconnaît le même langage. À partir de la relation d'équivalence entre états, on obtient l'automate quotient, c'est-à-dire l'automate dont les états sont les classes d'équivalence de l'automate initial et la fonction de transition est celle induite par la fonction de transition initiale compatible avec la relation d'équivalence.

Dans le chapitre 7, nous introduisons la notion d'automates non déterministes, en remplaçant dans la définition des automates la fonction de transition par une relation de transition : ainsi, à partir d'un état et d'un symbole, on peut atteindre un ensemble d'états au lieu d'un seul état. Nous avons la propriété suivante : pour tout langage reconnu par un automate non déterministe, il existe un automate reconnaissant le même langage. Nous en déduisons une procédure de déterminisation de l'automate non déterministe.

Dans le chapitre 8, nous introduisons les automates non déterministes avec ϵ -transition et on a un résultat similaire au précédent, c'est-à-dire que, à chacun de ces automates, on a un automate déterministe reconnaissant le même langage.

Dans le chapitre 9, nous introduisons les expressions régulières qui constituent un autre formalisme pour dénoter des langages d'états finis, ou langages rationnels.

Dans le chapitre 10, nous étudions le théorème de Kleene, indiquant que la classe des langages rationnels est celle des langages reconnus par des automates d'états finis. Ainsi, pour chaque expression régulière, nous pouvons trouver un automate reconnaissant le langage dénoté par cette expression régulière, et inversement.

Dans le chapitre 11, nous introduisons les grammaires et en particulier les grammaires régulières. Nous étudions l'équivalence entre formalismes pour le cas des grammaires régulières dans le chapitre 12 : pour tout langage reconnu par un automate à nombre fini d'états, il existe une expression régulière reconnaissant le même langage (et réciproquement), de plus il existe une grammaire régulière reconnaissant le même langage (et réciproquement).

Le chapitre 13 est dédié au lemme de l'itération, qui est une condition nécessaire à la régularité d'un langage. Nous utilisons le lemme de l'itération et les propriétés de fermeture

des langages réguliers dans le chapitre 14 pour démontrer que certains langages ne sont pas réguliers, c'est-à-dire qu'il n'existe pas d'automate reconnaissant ces langages.

Dans l'Annexe A, nous abordons la modélisation de problèmes et leur résolution en utilisant les automates. Nous introduisons également la notion de propriété caractérisée par un langage. Une modélisation (sous forme d'automate) satisfait une propriété si le langage associé à l'automate est inclus dans le langage associé à la propriété.

Remerciements

Cet ouvrage est né après plusieurs années d'enseignement d'un cours à l'Université Grenoble Alpes sur le sujet. Dans ses formes préliminaires, l'ouvrage a bénéficié du retour de plusieurs collègues participant à l'enseignement (en particulier Saddek Bensalem, Yassine Lakhnech et Anne Rasse), que nous remercions. Nous remercions également les étudiants, doctorants et post-doctorants de l'Université Grenoble Alpes Théo Barollet, Alaa Ben Fatma, Tom Cornebize, Florian Gallay, Ali Kassem, Charlotte Lefèvre et Marius Monnier, qui ont participé à la relecture des chapitres. Leurs relectures ont permis de s'assurer que les solutions des exercices soient abordables et claires du point de vue de l'étudiant. Enfin, nous remercions l'Université Grenoble Alpes¹, Inria Grenoble Rhône-Alpes² et le Laboratoire d'Informatique de Grenoble³ pour les moyens et l'environnement de qualité mis à notre disposition facilitant notre travail quotidien.

Pour contacter les auteurs

Pour vos remarques ou rapports de coquilles, vous pouvez contacter les auteurs par email⁴.

1. www.univ-grenoble-alpes.fr

2. www.inria.fr/fr/centre-inria-grenoble-rhone-alpes

3. www.liglab.fr

4. prenom.nom@univ-grenoble-alpes.fr

Introduction

Pourquoi étudier les automates et la théorie des langages ?

Cet ouvrage traite des concepts et techniques de langage à états, ou langage rationnel. Les notions abordées ont une importance fondamentale en informatique. Elles sont utilisées dans de nombreux domaines, que ce soit en informatique théorique ou appliquée. Le livre fait le lien entre une certaine classe de langages informatiques et les automates à nombre fini d'états (aussi appelés par ailleurs automates à états finis ou automates d'états finis). La notion d'automates est très ancienne⁵ et a été utilisée dans les années 1950-1960 pour le calcul. Les premiers processeurs étaient constitués d'une **partie contrôle** et d'une **partie opérative**. La partie contrôle est un automate qui pilote la partie opérative pour réaliser par exemple des opérations arithmétiques et logiques ou pour faire des transferts mémoires. Par la suite, les automates ont été utilisés dans :

- la conception de matériel informatique,
- la reconnaissance de mots, l'analyse lexicale d'un compilateur,
- la reconnaissance de texte (formulaire internet, moteur de recherche, etc.),
- la conception et la vérification de protocoles de communication,
- la programmation de robots, de planification,
- la modélisation de propriétés et la vérification de programmes,
- la compression de textes,
- l'intelligence artificielle.

Même si les automates sont très anciens et s'ils sont beaucoup utilisés en pratique de nos jours, ils constituent toujours un sujet étudié en recherche, en tant que tel ou comme outil de résolution de problème.

5. fr.wikipedia.org/wiki/Automate_mecanique

Première rencontre avec les automates

Dans ce qui suit, pour abréger, nous utilisons le terme automate pour automate à nombre fini d'états.

Un automate est représenté sous forme de graphe, dont les nœuds (cercles) représentent les états, les nœuds doubles les états accepteurs et les arcs (flèches) représentent les transitions ; voir figure 2. À l'intérieur du nœud, nous indiquons le nom de l'état. Un automate est caractérisé par un nombre fini d'états, un vocabulaire qui désigne des actions et des règles permettant de passer d'un état à un autre. Finalement, un automate est un quintuplet (ensemble d'états, un vocabulaire d'actions, un état initial, un nombre fini de règles de transition et un ensemble d'états accepteurs. D'autres automates, machines ne sont pas abordées dans ce livre : les automates à états finis avec vocabulaires d'entrée et de sortie comme les automates de Moore et de Mealy, les automates à piles en lien avec les grammaires hors contextes, les machines de Turing en lien avec les notions de calculabilité et de décidabilité.

Un langage est défini par un ensemble de symboles, de mots sur ces symboles et par une grammaire indiquant comment les mots peuvent être utilisés. Dans un compilateur, les mots du langage constituent un langage régulier et leur reconnaissance peut se faire par un automate à états finis.

Nous considérerons dans la suite quelques exemples d'automates.

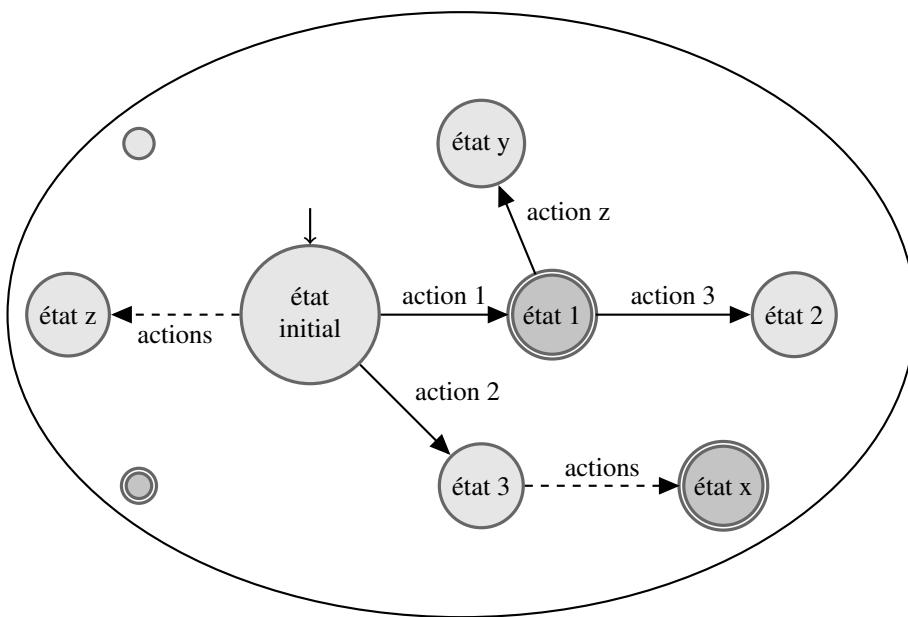


Figure 2 – Représentation schématique d'un automate.

Un automate reconnaissant la chaîne « abc »

Un premier exemple est un automate qui reconnaît une chaîne de caractères, mettons « abc ». Cet automate est représenté sur la figure 3. C'est un automate à quatre états : 0, 1, 2 et 3. Cet automate lit des mots (ou des phrases) sur l'alphabet latin. De l'état initial 0, par la lettre « a » on passe à l'état 1. De l'état 1, par « b », on va à l'état 2 et, de l'état 2, par la lettre « c », on va dans l'état 3, qui est un état accepteur. Tout ce qui n'est pas spécifié par l'automate conduit implicitement dans un état de rejet. Étant donné une séquence de symboles à lire, un automate lit chaque symbole, l'un après l'autre, dans l'ordre de la séquence. Au démarrage, avant de lire le premier symbole, un automate est dans son état initial.

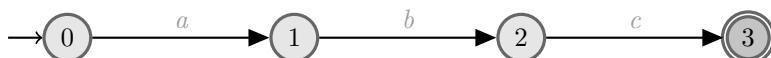


Figure 3 – Un automate lisant des mots sur l'alphabet latin et reconnaissant la chaîne « abc ».

Un automate reconnaissant les mots terminant par « ant »

Un second exemple d'automate est représenté dans la figure 4. Nous supposons pour simplifier que l'automate ne lit qu'un seul mot et doit déterminer si celui-ci termine par « ant », on dit dans ce cas que l'automate reconnaît ou accepte le mot. L'état initial rien vu est indiqué par une transition sans état de départ (flèche entrant dans le nœud rien vu). L'automate évolue d'état en état en fonction du symbole lu en entrée et de son état courant.

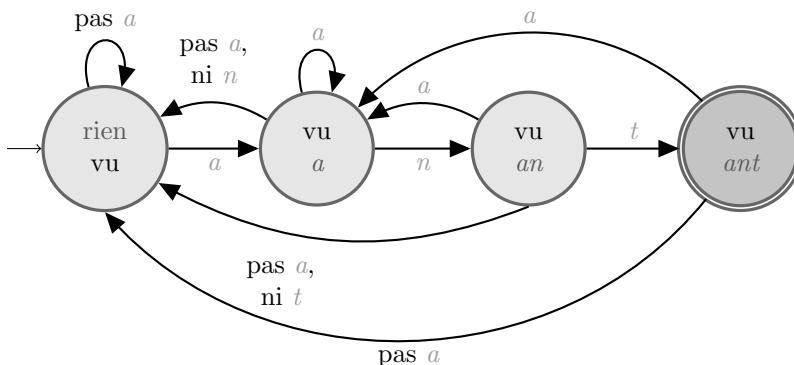


Figure 4 – Un automate lisant des mots sur l'alphabet latin et reconnaissant ceux qui terminent par « ant ».

- Depuis l'état rien vu, si le symbole lu est différent du symbole **a** alors l'automate reste dans l'état rien vu, sinon (si le caractère lu est **a**) alors l'automate va dans l'état vu **a**.
- Depuis l'état vu **a**, si le caractère lu est **a** alors l'automate reste dans l'état vu **a**, si le caractère lu est **n** alors l'automate va dans l'état vu **an**, sinon (si le caractère lu est ni **a** ni **n**), l'automate retourne dans l'état rien vu.

- Depuis l'état vu an, si le caractère lu est a alors l'automate retourne dans l'état vu a, si le caractère lu est t alors l'automate va dans l'état vu ant, sinon (si le caractère lu est ni a ni t), l'automate retourne dans l'état rien vu.
- Depuis l'état vu ant, si le caractère lu est a alors l'automate retourne dans l'état vu a, sinon (si le caractère lu n'est pas a) alors l'automate retourne dans l'état rien vu.

Si l'automate s'arrête dans l'état vu ant (qui est un état accepteur), alors l'automate accepte le mot. On remarque que les états de l'automate correspondent aux différentes situations dans la lecture caractère par caractère d'un mot par rapport au fait que ce mot termine par « ant ».

Des automates pour un protocole de transaction électronique

Nous modélisons une transaction électronique, avec comme entités un client, un marchand et une banque, inspiré de [10]; voir figure 5. Nous considérons un protocole très simplifié, mais qui nous permet d'illustrer sa vérification automatisée.

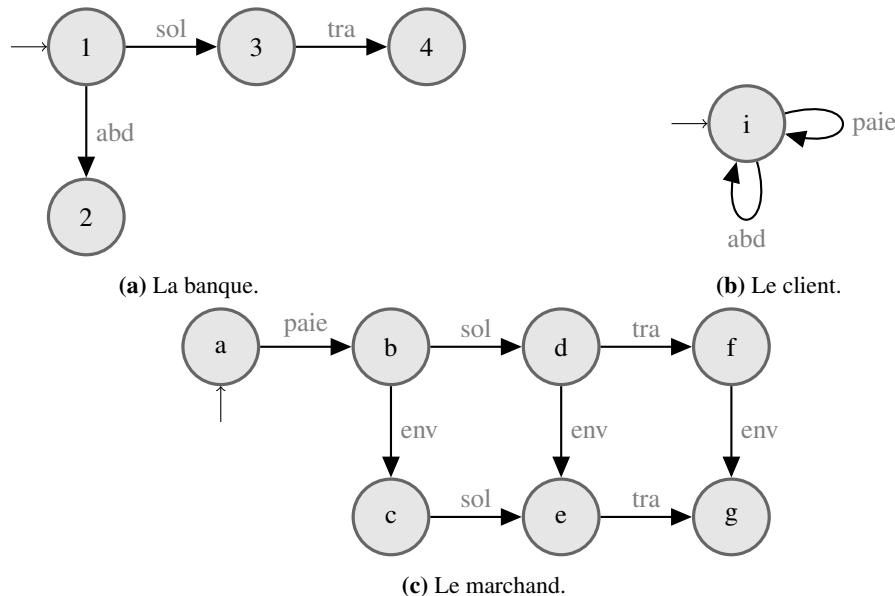


Figure 5 – Automates pour le protocole de transaction électronique.

Les participants de la transaction. Le client veut acheter une marchandise chez le marchand et la paye de manière électronique.

- Les actions du client sont « payer la marchandise » (action paie) ou « abandonner la transaction et récupérer son argent » (action abd).
- Le marchand peut envoyer la marchandise (action env) et solder le paiement (action sol). Ces deux actions peuvent être effectuées dans n'importe quel ordre.
- La banque peut transférer l'argent au marchand (action tra).

On suppose que la banque est honnête, que le marchand n'envoie la marchandise que s'il est payé. Par contre, on suppose que le client va tenter de recevoir la marchandise tout en récupérant son argent. On modélise le comportement de la banque, du client et du marchand par des automates ; voir les automates dans les figures 5a, 5b et 5c respectivement.

Comportement global (figure 6). Un état global du système est représenté par un triplet (état1 , état2 , état3), état1 représentant l'état de la banque, état2 celui du client et état3 celui du marchand. L'état initial étant (1, i, a). L'évolution globale du système peut être décrite par un automate composé soit en synchronisant les actions communes à au moins deux participants, soit en évoluant d'un état à l'autre lorsqu'une action n'est présente que dans un seul automate. Les transitions globales sont obtenues comme suit :

- soit une action n'est possible que dans seul automate composant (par exemple env n'est possible que chez le marchand et celle-ci n'est possible que lorsque l'état global est (e1, e2, e3) avec e3 qui peut être b, d ou f;
- soit l'action est possible dans deux automates et l'évolution globale est possible uniquement si elle est possible dans chacun des deux automates.

Ainsi de l'état (1, i, a) on peut aller

- par l'abd dans (2, i, a) ou
- par paie dans l'état (1, i, b) puis par env dans l'état (1, i, c) puis par sol dans l'état (3, i, e) puis par tra dans l'état (4, i, g).

Nous obtenons le diagramme représenté dans la figure 6. Les états encadrés sont ceux où le protocole ne peut plus évoluer :

- dans la configuration (4, i, g), la banque et le marchand ont terminé leur exécution et le client ne peut plus exécuter les actions paie et abd qui sont synchronisées;
- dans la configuration (2, i, c), la banque a terminé son exécution et le marchand ne peut exécuter l'action sol qui est synchronisée avec l'action sol de la banque qui, elle, n'est pas dans un état où elle peut exécuter cette action.

Au final, nous avons construit le comportement global du protocole compatible avec les comportements individuels des participants. Pour obtenir le comportement global, nous avons directement utilisé les comportements des participants que nous avons synchronisés. Ce comportement global peut être obtenu de manière systématique et automatique en utilisant la construction de l'automate produit que nous étudierons au chapitre 4. Plus généralement, étant donnés n automates, l'automate composé est un automate dont les états sont des n -uplets d'états locaux et les transitions sont de deux sortes :

- soit une action n'apparaît que dans un automate et la transition globale n'est possible dans l'état global que si elle est possible dans l'état local correspondant;
- soit une action apparaît dans plusieurs automates et la transition globale n'est possible que si elle l'est à partir de chacun des automates locaux ; dans ce cas, on dit que les automates se synchronisent sur l'action en question.

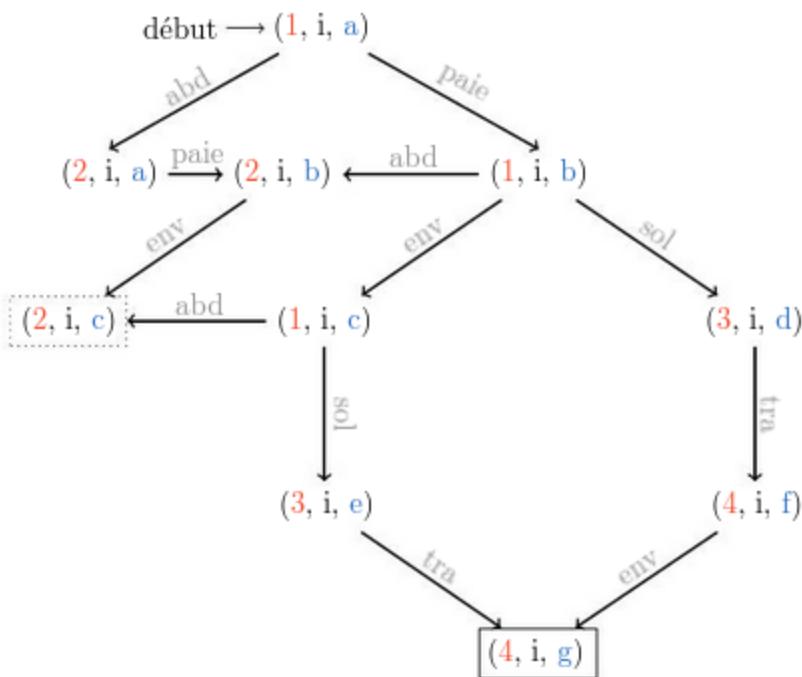


Figure 6 – Comportement global du protocole. Un état global est représenté par un triplet (état1, état2, état3), état1 représentant l'état de la banque, état2 celui du client et état3 celui du marchand. Les transitions entre états représentent l'évolution de l'état global du protocole.

Propriétés du protocole. Avec le comportement global du protocole, qui correspond aux interactions pouvant réellement se produire entre les participants, nous pouvons vérifier si celui-ci fonctionne correctement et/ou est sécurisé. Par exemple, nous pouvons nous poser les questions suivantes :

- Est-ce que toute transaction initiée se termine soit par un abandon, soit une réalisation de la transaction ? Autrement dit, est-ce que toute exécution du protocole se termine normalement et ne se retrouve pas bloquée ?
- Est-il possible que le client récupère la marchandise sans payer le marchand ?

Malheureusement, pour le protocole que nous considérons ici, la réponse à la première question est négative et celle à la seconde positive. En effet, dans le comportement global, on peut voir qu'il existe la séquence d'actions *paie*, *env*; *abd* faisant passer par les configurations (1, i, a), (1, i, b), (1, i, c), (2, i, c) où, d'une part, le protocole ne peut plus évoluer (le marchand est coincé) et, d'autre part, le marchand a envoyé la marchandise sans que le client ait payé.

Sur cet exemple simple de transaction, nous pouvons déterminer les réponses à ces questions en examinant le comportement des participants et le comportement global. Cependant, sur des protocoles réalistes, les automates modèles sont bien évidemment de taille bien plus importante et il n'est pas envisageable de les analyser « manuellement » de manière précise. Il s'agit alors de pouvoir répondre à de telles questions de *manière automatique* en utilisant des algorithmes et des programmes informatiques les implémentant. C'est l'objectif de certains algorithmes étudiés dans ce livre qui sont mis en application dans le chapitre annexe.

Rappels et notations

Ce chapitre contient un rappel des notions mathématiques nécessaires et suffisantes pour l'étude des concepts abordés dans ce livre. Il ne saurait se substituer aux cours de mathématiques sur les notions abordées. Le lecteur pourra consulter en complément l'excellent M@ths en LIGne¹ de Bernard Ycart ou les références [20, 22].

1.1 Notions de logique, assertions

Définition 1 (Assertion (logique)) *On appelle assertion (logique) tout énoncé mathématique susceptible d'être vrai ou faux.*

Lorsqu'une assertion dépend de plusieurs sous-assertions, il est possible d'utiliser une table de vérité pour définir la valeur de vérité de l'assertion en fonction des *valeurs de vérité*, vrai et faux (abrévés V et F), des assertions dont elle dépend.

Tableau 1.1 – Tables de vérité des opérateurs booléens.

		$P \wedge Q$		$P \vee Q$	
		P	Q	P	Q
P	$\neg P$	V	V	V	V
		V	F	F	F
V	F	F	V	F	V
F	V	F	F	F	V
		F	F	F	F

Définition 2 (Table de vérité) Une table de vérité est un tableau où sur la première ligne sont indiqués d'abord à gauche des noms d'assertions, puis en fin de ligne l'assertion définie par la table de vérité (qui dépend des assertions précédentes). Sur les lignes suivantes, on

1. <https://lik.imaq.fr/membres/Bernard.Ycart/mel/>

représente chaque combinaison des valeurs de vérité des assertions déterminant la valeur de vérité de l'assertion définie. Sur chaque ligne, à la position de la dernière colonne, on indique la valeur de vérité de l'assertion définie en fonction des valeurs de vérité des autres assertions indiquées sur cette même ligne.

Dans la suite de cette section, nous considérons deux assertions P et Q .

Définition 3 (Équivalence d'assertions) *Les assertions P et Q sont équivalentes si elles ont les mêmes tables de vérité, c'est-à-dire si elles sont soit vraies en même temps soit fausses en même temps.*

Définition 4 (Opérateurs booléens) *L'application d'opérateurs booléens à une assertion définit une nouvelle assertion. Plus précisément :*

- la négation de P est l'assertion notée $\text{non}(P) \sim P$;
 - la conjonction de P et Q est l'assertion notée P et Q , aussi notée $P \wedge Q$;
 - la disjonction de P et Q est l'assertion notée P ou Q , aussi notée $P \vee Q$;
- définies selon les tables de vérités représentées dans le tableau 1.1.

Définition 5 (Opérateurs dérivés, implication et équivalence) *Nous définissons les opérateurs d'implication et d'équivalence, dérivés des opérateurs booléens.*

- On dit que l'assertion P implique l'assertion Q , lorsque l'assertion Q est vraie dès que P l'est. Une définition plus explicite est donnée par le tableau 1.2a.
- On dit que l'assertion P est équivalente à l'assertion Q , lorsque l'assertion Q est vraie dès que P l'est et que l'assertion P est vraie dès lors que Q l'est. Une définition plus explicite est donnée par le tableau 1.2b.

Tableau 1.2 – Tables de vérité de l'implication et de l'équivalence.

P	Q	$P \implies Q$
V	V	V
V	F	F
F	V	V
F	F	V

(a) Implication.

P	Q	$P \iff Q$
V	V	V
V	F	F
F	V	F
F	F	V

(b) Équivalence.

Dans la suite, pour simplifier l'exposé, nous ne ferons pas la distinction entre la notion d'équivalence présentée dans la définition 3 et celle présentée dans la définition 5.

On notera (en utilisant les tables de vérité) que $P \implies Q$ est équivalent à $(\neg P) \vee Q$ et que $P \iff Q$ est équivalent à $(P \wedge Q) \vee ((\neg P) \wedge (\neg Q))$.

Définition 6 (Ou exclusif) *L'application de l'opérateur de « ou exclusif » entre P et Q , noté P xor Q , est défini comme l'assertion $(P \wedge (\neg Q)) \vee ((\neg P) \wedge Q)$.*

Pour alléger l'écriture en omettant les parenthèses, on utilise les priorités des opérateurs les uns par rapport aux autres : on a que \neg est plus prioritaire que \wedge qui est plus prioritaire que \vee .

Propriété 1 (Propriétés des opérateurs booléens) *L'opérateur de conjonction (resp. disjonction) est commutatif, associatif, d'élément neutre V (resp. F), idempotent, et d'élément absorbant F (resp. V). Ces propriétés sont indiquées dans le tableau 1.3. D'autres propriétés comme la distributivité peuvent être montrées en utilisant les tables de vérité.*

Tableau 1.3 – Certaines propriétés des opérateurs booléens.

propriété	équivalence associée	
associativité	$P \wedge (Q \wedge R)$	$\iff (P \wedge Q) \wedge R$
commutativité	$P \wedge Q$	$\iff Q \wedge P$
V élément neutre pour \wedge	$P \wedge V$	$\iff P$
F élément neutre pour \vee	$P \vee F$	$\iff P$
V élément absorbant pour \vee	$P \vee V$	$\iff V$
F élément absorbant pour \wedge	$P \wedge F$	$\iff F$
idempotence de \wedge	$P \wedge P$	$\iff P$
idempotence de \vee	$P \vee P$	$\iff P$
distributivité de \wedge par rapport à \vee	$P \wedge (Q \vee R)$	$\iff (P \wedge Q) \vee (P \wedge R)$
distributivité de \vee par rapport à \wedge	$P \vee (Q \wedge R)$	$\iff (P \vee Q) \wedge (P \vee R)$

Définition 7 (Condition nécessaire et condition suffisante) *Q est une condition nécessaire à P si $P \implies Q$. Q est une condition suffisante à P si $Q \implies P$.*

1.2 Quantificateurs

Nous avons vu qu'une assertion est un énoncé dont on peut dire s'il est vrai ou faux. Si on considère un énoncé qui dépend d'un paramètre x , que l'on note $P(x)$, l'évaluation de cet énoncé (comme étant vrai ou faux) dépend de la valeur de ce paramètre.

Définition 8 (Prédicat) *On appelle prédicat un énoncé mathématique dont la valeur de vérité dépend de la valeur de ses paramètres.*

Définition 9 (Quantification universelle) *Le prédicat $\forall x . P(x)$ est évalué à vrai dès lors que $P(x)$ est vrai quelque soit la valeur du paramètre x . L'opérateur \forall est dit de quantification universelle.*

Définition 10 (Quantification existentielle) *Le prédicat $\exists x . P(x)$ est évalué à vrai dès lors que $P(x)$ est vrai pour (au moins) une valeur du paramètre x . L'opérateur \exists est dit de quantification existentielle.*

En utilisant les ensembles (voir section suivante), les quantifications peuvent être exprimées par rapport à un ensemble.

1.3 Notions ensemblistes

Définition 11 (Ensemble) *On appelle ensemble toute collection non ambiguë d'objets distincts deux à deux, appelés éléments de l'ensemble. L'ensemble vide, noté \emptyset , est l'ensemble ne contenant aucun élément.*

Dans la suite, nous considérons trois ensembles E , F et G .

Définition 12 (Appartenance) *Le fait qu'un élément x appartienne à un ensemble E se note $x \in E$, et son contraire $x \notin E$ (x n'appartient pas à E).*

Il existe plusieurs manières de définir un ensemble :

- en *extension*, les éléments de l'ensemble sont énumérés (donnés un à un) ;
- en *intension*, les éléments de l'ensemble sont ceux satisfaisant un prédictat, typiquement de la forme $\{x \mid P(x)\}$ qui est l'ensemble des x tels que $P(x)$ est vrai.

Définition 13 (Inclusion d'un ensemble dans un autre) *E est inclus dans l'ensemble F , noté $E \subseteq F$, si tous les éléments de l'ensemble E appartiennent à l'ensemble F :*

$$\forall x. (x \in E \implies x \in F).$$

Ou, de manière équivalente : $\forall x \in E. x \in F$.

On dit que E est une partie de F , ou encore que E est un sous-ensemble de F . On note $E \subset F$ lorsque E est strictement inclus dans F , c'est-à-dire lorsque E est inclus dans F mais que F n'est pas inclus dans E .

Définition 14 (Égalité) *E et F sont égaux, noté $E = F$, si $E \subseteq F$ et $F \subseteq E$.*

Définition 15 (Non-égalité) *E et F ne sont pas égaux, noté $E \neq F$, si $\neg(E = F)$. On dit aussi que E et F sont distincts.*

Définition 16 (Ensemble des parties d'un ensemble) *L'ensemble des parties de E , noté $\mathcal{P}(E)$ ou 2^E , est l'ensemble contenant tous les sous-ensembles de E , c'est-à-dire l'ensemble $\{X \mid X \subseteq E\}$.*

Définition 17 (Complémentaire d'un ensemble) *Le complémentaire de E par rapport à F , noté $\complement_F E$ ou \overline{E} lorsque le contexte est clair, est l'ensemble défini comme suit :*

$$\overline{E} = \{x \mid x \notin E\}.$$

Définition 18 (Union d'ensembles) *L'union de E et F , notée $E \cup F$, est l'ensemble défini comme suit :*

$$E \cup F = \{x \mid x \in E \vee x \in F\}.$$

Définition 19 (Intersection d'ensembles) *L'intersection de E et F , notée $E \cap F$, est l'ensemble défini comme suit :*

$$E \cap F = \{x \mid x \in E \wedge x \in F\}.$$

Propriété 2 (Propriétés des opérateurs ensemblistes) Les opérateurs d'union et d'intersection sont associatifs, commutatifs et distributifs l'un par rapport à l'autre. L'ensemble vide \emptyset est élément neutre pour l'union et l'élément absorbant pour l'intersection.

Ces propriétés découlent des propriétés des opérateurs booléens².

Définition 20 (Couple et produit cartésien) Étant donnés deux éléments $x \in E$ et $y \in F$, le couple formé par x et y , noté (x, y) , est l'ensemble $\{(x), (x, y)\}$ à deux éléments. Le produit cartésien de E et F , noté $E \times F$, est l'ensemble $\{(x, y) \mid x \in E \wedge y \in F\}$.

Définition 21 (Relation) Une relation R entre E et F est un sous-ensemble du produit cartésien $E \times F : R \subseteq E \times F$. Pour une relation R entre E et F et deux éléments $e \in E$ et $f \in F$, lorsque e et f sont en relation selon R , on note $(e, f) \in R$ ou eRf .

Lorsque $E = F$, $R \subseteq E \times E$ est une relation sur E .

Définition 22 (Propriétés remarquables des relations) Soit R une relation sur E .

- R est réflexive si $\forall x . x \in E \implies (x, x) \in R$.
- R est antiréflexive si $\forall x . x \in E \implies (x, x) \notin R$.
- R est symétrique si $\forall x, y . (x, y) \in R \implies (y, x) \in R$.
- R est antisymétrique si $\forall x, y . ((x, y) \in R \wedge (y, x) \in R \implies x = y)$.
- R est transitive si $\forall x, y, z . ((x, y) \in R \wedge (y, z) \in R \implies (x, z) \in R)$.
- R est une relation d'équivalence si elle est réflexive, symétrique et transitive.
- R est une relation d'ordre si elle est réflexive, antisymétrique et transitive.

Définition 23 (Composition de relations) Soient $R \subseteq E \times F$ et $S \subseteq F \times G$ deux relations, la composition de R et S , notée $R \circ S$, est définie comme suit :

$$R \circ S = \{(x, z) \mid \exists y \in F . ((x, y) \in R \wedge (y, z) \in S)\}.$$

Définition 24 (Fermeture transitive) Soit $R \subseteq E \times E$ une relation sur E et n un entier positif. On note $R^1 = R$ et $R^n = R^{n-1} \circ R$. La fermeture transitive est la relation $R^+ = \bigcup_{n>0} R^n$.

Définition 25 (Fermeture réflexive et transitive) Soit E un ensemble, on note la relation identité $I = \{(x, x) \mid x \in E\}$; la fermeture réflexive et transitive de R est la relation $R^* = I \cup R^+$.

Définition 26 (Classe d'équivalence, partition et ensemble quotient) Soit R une relation d'équivalence sur E , la classe d'équivalence d'un élément $x \in E$, notée $[x]_R$, est définie comme étant l'ensemble $\{y \mid (x, y) \in R\}$ inclus dans E . L'ensemble des classes d'équivalence forme une partition de E , c'est-à-dire un ensemble dont les éléments sont deux à deux disjoints ($\forall x, y . x \neq y \implies ([x]_R \cap [y]_R = \emptyset)$) et tel que $\bigcup_{x \in E} [x]_R = E$.

2. Soit \mathcal{A} un ensemble d'assertions, $(\mathcal{A}, \wedge, \vee, \neg, \top)$ est une algèbre booléenne. À l'ensemble E , nous pouvons associer l'algèbre booléenne $(2^E, \cup, \cap, E, \emptyset)$.

Inversement, toute partition \mathcal{P} de E induit une relation d'équivalence R sur E :

$$(x, y) \in R \quad \text{ssi} \quad \exists C \in \mathcal{P}. \ x, y \in C$$

L'ensemble des classes d'équivalence $\{[x]_R \mid x \in E\}$ est appelé ensemble quotient de E par R .

Définition 27 (Fonction et application) Une fonction de E vers F fait correspondre à un élément x de E un unique élément $f(x)$ de F . Elle est notée $f : E \rightarrow F$ ou $x \mapsto f(x)$. Ou encore si $\Gamma = \{(x, f(x)) \mid x \in E\}$, alors $\forall x \in E. \forall y \in F. ((x, y) \in \Gamma \wedge (x, z) \in \Gamma \implies y = z)$; Γ est le graphe de la fonction. Lorsqu'une fonction f fait correspondre à un élément x un élément y , nous le notons $y = f(x)$ ou $(x, y) \in f$ en fonction du contexte.

On dit que la fonction f est totale (ou est une application) si $f(x)$ est définie pour tout $x \in E$. Une fonction partielle est une fonction non totale.

On note $E \mapsto F$ ou F^E l'ensemble des fonctions de E vers F .

Définition 28 (Point fixe) Un point fixe d'une fonction $f : E \rightarrow E$ est un élément $x \in E$ tel que $f(x) = x$.

Définition 29 (Domaine et codomaine) Le domaine d'une fonction $f : E \rightarrow F$ est le sous-ensemble de E tel que f est définie, son codomaine est le sous-ensemble de F dont les éléments sont des images par f d'un élément de E :

- domaine(f) = $\{x \in E \mid f(x) \text{ est définie}\}$ et
- codomaine(f) = $\{y \in F \mid \exists x \in E. f(x) = y\}$.

Définition 30 (Ensemble ordonné) Un ensemble ordonné est la donnée d'un couple (E, R) où R est une relation d'ordre sur E . On dit que (E, R) est totalement ordonné si $\forall x, y \in E. (x, y) \in R \vee (y, x) \in R$. Dans le cas contraire, on dit que (E, R) est partiellement ordonné.

Définition 31 (Suite ou séquence) Soit E un ensemble, une suite ou une séquence d'éléments de E est une application de \mathbb{N} ou d'un intervalle de \mathbb{N} commençant à 0 ou à 1 vers E .

Définition 32 (Propriétés d'injection, surjection et bijection) Soit f une fonction de E vers F .

- f est injective si $\forall x, y \in E. f(x) = f(y) \implies x = y$.
- f est surjective si $\forall y \in F. \exists x \in E. f(x) = y$.
- f est bijective si elle est à la fois injective et surjective.

Théorème 1 (Principe des tiroirs) Supposons E et F finis et tels que $|E| > |F|$. Soit f une application de E vers F . Il n'existe pas de fonction injective de E vers F . Autrement, au moins une image de f (dans F) possède deux antécédents (dans E).

Le principe des tiroirs est aussi appelé principe de Dirichlet (1834); en anglais « pigeonhole principle ».

Définition 33 (Propriété de croissance) Soient (E, R) et (F, S) deux ensembles ordonnés. Une fonction de E vers F est croissante si $\forall x, y. (x, y) \in R \implies (f(x), f(y)) \in S$.

Définition 34 (Loi de composition interne) Une loi de composition interne sur l'ensemble E est une application de $E \times E \mapsto E$.

Dans les deux définitions suivantes, nous considérons un ensemble E muni d'une loi de composition interne notée $*$.

Définition 35 (Monoïde) Un monoïde est un triplet $(E, *, e)$ est un monoïde où :

- E est un ensemble,
- $*$ est une loi de composition interne associative : $\forall x, y, z \in E. (x * y) * z = x * (y * z)$,
- il existe un élément neutre e pour $*$: $\exists e \in E. \forall x \in E. e * x = x * e = x$.

Définition 36 (Groupe) Un groupe est un monoïde $(E, *, e)$ si chaque élément de E admet un élément symétrique : $\forall x \in E. \exists y \in E. x * y = y * x = e$. De plus, si $*$ est commutative ($\forall x \in E. \forall y \in E. x * y = y * x$), on dit que le groupe est commutatif ou abélien.

Définition 37 (Morphisme) Soient (G, \bullet, e_G) et $(G', *, e_{G'})$ deux groupes. Une application $f : G \rightarrow G'$ est un morphisme de groupe si $\forall x, y \in G. f(x \bullet y) = f(x) * f(y)$.

1.4 Entiers naturels

Définition 38 (Entiers naturels et relatifs) On note \mathbb{N} l'ensemble des entiers naturels $0, 1, 2, \dots$ et \mathbb{Z} l'ensemble des entiers relatifs $\dots, -2, -1, 0, 1, 2, \dots$ On note \mathbb{N}^* l'ensemble $\mathbb{N} \setminus \{0\}$. La relation d'ordre usuelle entre deux entiers m et n est notée $m \leq n$ et est définie par $\exists h \in \mathbb{N}. n = m + h$; c'est une relation d'ordre totale. L'intervalle sur \mathbb{N} entre deux entiers a et b avec $a \leq b$, noté $[a, b]$, est l'ensemble $\{x \in \mathbb{N} \mid a \leq x \wedge x \leq b\}$.

Définition 39 (Cardinal d'un ensemble) Un ensemble non vide est de cardinal $n \in \mathbb{N}^*$, s'il est en bijection avec $[1, n]$, c'est-à-dire s'il contient exactement n éléments. L'ensemble vide est de cardinal 0. Deux ensembles ont le même cardinal s'ils sont en bijection.

Définition 40 (Ensembles équivalents, finis, infinis, dénombrables)

- Deux ensembles E et F sont équivalents s'il existe une bijection de E dans F .
- Un ensemble est fini s'il existe un entier naturel $n \in \mathbb{N}$ qui soit le cardinal de cet ensemble.
- Deux ensembles finis sont équivalents s'ils ont le même cardinal.
- Un ensemble est infini s'il n'est pas fini, comme \mathbb{N} par exemple.
- Un ensemble est dénombrable s'il est équivalent à \mathbb{N} .

Définition 41 (Représentation des entiers) Soit b un chiffre. Tout entier naturel $x \in \mathbb{N}$ peut s'écrire $x = x_n \times b^n + \dots + x_1 \times b + x_0$, où les b^i sont les entiers correspondant aux $i^{\text{ème}}$ puissances de b (interprété comme un entier suivant l'interprétation standard des chiffres) et $\forall i \in [0, n]. x_i \in [0, b - 1]$. L'entier x peut alors être représenté par $x_n x_{n-1} \dots x_0$ et le chiffre b est appelé base de représentation de x .

Dans la suite, nous considérons trois entiers naturels $a, b, n \in \mathbb{N}$.

Définition 42 (Divisibilité) a est divisible par b s'il existe un entier naturel $k \in \mathbb{N}$ tel que $a = b \times k$.

Définition 43 (Congruence) a est congru à b modulo n , noté $a \equiv b[n]$, si $a - b$ est divisible par n .

Cette congruence est une relation d'équivalence sur l'ensemble des entiers relatifs.

Propriété 3 (Congruences) Supposons $a \equiv b[n]$ et $c \equiv d[n]$ et soit x un entier relatif. On a :

$$\begin{aligned} a + c &\equiv (b + d)[n], \\ a \times c &\equiv (b \times d)[n], \\ a \times x &\equiv (b \times x)[n]. \end{aligned}$$

Définition 44 (Classes d'équivalence d'un entier) Étant donné un entier $n \in \mathbb{N}$, on a les classes d'équivalence suivantes, notée $[k]$ pour $k \in [0, n - 1]$: $[k] = \{k + n \times q \mid q \in \mathbb{N}\}$.

1.5 Ensembles définis inductivement

Soit U un ensemble.

Définition 45 (Ensemble défini inductivement) Une définition inductive d'un sous-ensemble E de U consiste en la donnée :

- d'un ensemble $B \subseteq U$ contenant certains éléments de U , appelé base,
- d'un ensemble R de fonctions partielles $f : U^n \rightarrow U$ où n est l'arité de f (c'est-à-dire le nombre d'arguments de f), appelé ensemble des règles de construction.

Ainsi, l'ensemble E est le plus petit ensemble vérifiant :

- (base) $B \subseteq E$,
- (induction) $\forall f \in R. \forall x_1, \dots, x_n \in E. f(x_1, \dots, x_n) \in E$.

On dit aussi que E est défini par induction.

Théorème 2 Si E est défini inductivement par (B, R) alors $X = \bigcup_{i \in \mathbb{N}} X_i$, avec :

- $X_0 = B$,
- $X_{i+1} = X_i \cup \{f(x_1, \dots, x_n) \mid x_i \in X_i \wedge f \in R\}$.

1.6 Raisonnements, techniques de preuve

Dans cette section, nous considérons deux assertions ou prédictats P et Q .

Définition 46 (Preuve en utilisant le principe du tiers exclu) On cherche à démontrer P , pour cela on démontre que P est vraie sous l'hypothèse H et qu'elle est vraie sous l'hypothèse $\neg H$.

Définition 47 (Preuve par l'absurde) Il y a deux variantes de la technique en fonction de ce que l'on souhaite démontrer.

- On cherche à démontrer P , pour cela on trouve un couple $(Q, \neg Q)$ d'assertions contradictoires et on démontre $\neg P \implies Q$ et $\neg P \implies \neg Q$.
- On cherche à démontrer $P \implies Q$, pour cela on suppose $P \wedge \neg Q$ et on déduit une contradiction.

Définition 48 (Preuve par contraposition) On cherche à démontrer $P \implies Q$, pour cela on démontre $\neg Q \implies \neg P$, qui est appelée la (proposition) contraposée.

Définition 49 (Preuve par récurrence) On cherche à démontrer qu'un prédicat $P(n)$ est vrai pour tout $n \in \mathbb{N}$, c'est-à-dire $\forall n \in \mathbb{N}. P(n)$, pour cela on peut utiliser une récurrence (faible) ou une récurrence forte.

Dans le cas de la récurrence (faible) :

- (cas de base) on montre $P(0)$,
- (pas de récurrence) on montre $\forall n \in \mathbb{N}. P(n) \implies P(n+1)$.

Le pas de récurrence consiste à montrer que le prédicat reste vrai d'un rang à l'autre.

Dans le cas de la récurrence forte, on montre :

- (cas de base) $P(0)$,
- (pas de récurrence) $\forall n \in \mathbb{N}. (\forall k. k \leq n \implies P(k)) \implies P(n+1)$.

Le pas de récurrence consiste à montrer que si le prédicat est vrai jusqu'à un certain rang, alors il reste vrai au rang suivant.

Remarque 1 Si on cherche à montrer que le prédicat $P(n)$ est vrai sur un intervalle ouvert $[a, \infty[$, à partir d'un entier a :

- en utilisant la récurrence faible, on montre $P(a)$ et $\forall n \in [a, \infty[. P(n) \implies P(n+1)$;
- en utilisant la récurrence forte, on montre $P(a)$ et $\forall n \in [a, \infty[. (\forall k. k \leq n \implies P(k)) \implies P(n+1)$.

Définition 50 (Preuve par induction) Soit E un ensemble défini par induction par (B, R) . On cherche à démontrer qu'un prédicat $P(e)$ est vrai pour tout e dans E , c'est-à-dire $\forall e \in E. P(e)$, pour cela on montre :

- (cas de base) $\forall b \in B. P(b)$,
- (pas d'induction)

$$\forall f \in R \text{ (d'arité } k\text{). } \forall x_1, \dots, x_k \in E. (\forall x_i. P(x_i)) \implies P(f(x_1, \dots, x_k)).$$

Le pas d'induction consiste à montrer que si le prédicat est vrai sur un élément, alors il reste vrai sur tous les éléments que l'on peut obtenir avec les règles de construction.



Chapitre 2

Notions préliminaires

2.1 Résumé intuitif du chapitre

Dans ce chapitre, nous introduisons les notions de bases des automates. Plus généralement, les notions présentées ci-après sont les concepts de base de la théorie des langages (réguliers) et des automates. Nous introduisons les notions d'**alphabet**, **mot**, **concaténation** et **langage**.

Nous verrons que les automates peuvent être vus comme des machines, lisant des entrées. Un automate est toujours prêt pour recevoir une entrée. L'ensemble des entrées qu'un automate peut recevoir s'appelle son alphabet. Un alphabet est un ensemble fini dont les éléments sont appelés **lettres** ou **symboles**. L'intuition est que lorsqu'on soumet un symbole à un automate, celui-ci réagit uniquement si le symbole fait partie de son alphabet et l'ignore autrement.

En soumettant plusieurs symboles, les uns après les autres, la séquence de symboles forme un mot. Un mot est donc une séquence de symboles, dans un ordre bien précis.

Deux mots peuvent être composés. Par exemple, une opération fréquente de composition de mots est la concaténation. La concaténation est une opération qui permet, étant donnés deux mots, de construire un mot plus grand qui est le mot formé par les symboles du premier mot et dont les symboles suivants sont les symboles du deuxième mot.

Les automates que nous allons étudier sont principalement dédiés à la *reconnaissance de mots*. C'est-à-dire que lorsqu'on lui soumet une séquence de symbole (c'est-à-dire un mot) en entrée, un automate indique (par un mécanisme que nous étudierons au chapitre 3) s'il reconnaît le mot. Le langage associé à un automate est l'ensemble des mots (que l'on peut construire sur un alphabet donné) que l'automate reconnaît. Nous verrons que la classe des langages reconnus par un automate est la classe des langages réguliers (ou rationnels).

2.2 Les notions essentielles

Définition 51 (Alphabet) *Un alphabet Σ est un ensemble fini dont les éléments sont appelés symboles.*

Définition 52 (Mot) *Un mot de longueur $n \in \mathbb{N}$ est une application de $[1, n]$ vers Σ .*

On note $|u|$ la *longueur* du mot u . Pour $s \in \Sigma$, on note $|u|_s$ le nombre d'occurrences du symbole s dans u . Le *mot vide*, noté ϵ_Σ ou ϵ quand l'alphabet se déduit du contexte, est la fonction de l'ensemble vide (\emptyset) vers Σ .

Définition 53 (Mot, construit inductivement par la droite) *Considérons un alphabet Σ .*

- ϵ est un mot sur Σ ,
- si u est un mot sur Σ et s un symbole de Σ , alors $u \cdot s$ est un mot sur Σ .

La longueur d'un mot est le nombre de symboles apparaissant dans ce mot.

Définition 54 (Mot, construit inductivement par la gauche) *Considérons un alphabet Σ .*

- ϵ est un mot sur Σ ,
- si u est un mot sur Σ et s un symbole de Σ , alors $s \cdot u$ est un mot sur Σ .

La longueur d'un mot est le nombre de symboles apparaissant dans ce mot.

Remarque 2 *Dans les exercices, nous verrons que les définitions de mots sont équivalentes.*

Définition 55 (Langage) *Un langage sur l'alphabet Σ est un ensemble de mots sur Σ .*

Dans la suite, nous considérons des langages L , L_1 et L_2 .

Définition 56 (Langage universel) *Le langage universel sur l'alphabet Σ est l'ensemble de tous les mots que l'on peut définir sur Σ ; il est noté Σ^* . L'ensemble de tous les mots de longueur k sur Σ est noté Σ^k . L'ensemble de tous les mots de longueur au plus k sur Σ est noté $\Sigma^{\leq k}$.*

Définition 57 (Concaténation de mots) *Nous considérons la définition de mot sous forme d'application (définition 52). La concaténation de mots est une application de $\Sigma^* \times \Sigma^*$ vers Σ^* . La concaténation de deux mots u et v dans Σ^* , notée $u \odot v$, est définie comme le mot $u \odot v : [1, |u| + |v|] \rightarrow \Sigma$ tel que :*

$$(u \odot v)(i) = \begin{cases} u(i) & \text{si } i \in [1, |u|], \\ v(i - |u|) & \text{si } i \in [|u| + 1, |u| + |v|]. \end{cases}$$

Remarque 3 (Définition de la concaténation pour les autres définitions de mot) *Les opérations de concaténation pour les mots définis inductivement par la droite et la gauche font l'objet de l'exercice 5.*

Définition 58 (Préfixe, suffixe et facteur) *Considérons deux mots u et v sur un alphabet Σ .*

- v est un préfixe de u , noté $v \preceq u$, s'il existe un mot v' (sur Σ) tel que $v \odot v' = u$.
- v est un suffixe de u , s'il existe un mot v' (sur Σ) tel que $v' \odot v = u$.
- v est un facteur de u , s'il existe deux mots v' et v'' (sur Σ) tels que $v' \odot v \odot v'' = u$.
- v est une extension de u , si u est un préfixe de v .

Remarque 4 *Le mot vide ϵ est préfixe, suffixe et facteur de tout mot.*

Considérons un mot u sur un alphabet Σ tel que $|u| \geq n$. Le préfixe de u jusqu'à l'élément d'indice n est noté $u_{\dots n}$; ce préfixe est de longueur n . Le suffixe de u à partir de l'élément d'indice n est noté $u_{n\dots}$: ce suffixe est de longueur $|u| - n + 1$.

Définition 59 (Concaténation de langages) La concaténation de langages est une application de $\mathcal{P}(\Sigma^*) \times \mathcal{P}(\Sigma^*)$ vers $\mathcal{P}(\Sigma^*)$. La concaténation de L_1 et L_2 , notée $L_1 \odot L_2$, est définie comme le langage $\{u_1 \odot u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\}$.

Définition 60 (Fermeture par préfixe, suffixe, facteur et extension) Ces fermetures sont définies pour des langages comme suit :

- La fermeture par préfixe de L , notée $\text{Pref}(L)$, est le langage formé par l'ensemble des préfixes des mots de L , défini comme :

$$\text{Pref}(L) = \{w \in \Sigma^* \mid \exists w' \in \Sigma^*. w \odot w' \in L\} ;$$

de manière équivalente : $\text{Pref}(L) = \{w \in \Sigma^* \mid \exists w' \in L. w \preceq w'\}$.

- La fermeture par suffixe de L , notée $\text{Suf}(L)$, est le langage formé par l'ensemble des suffixes des mots de L , défini comme :

$$\text{Suf}(L) = \{w \in \Sigma^* \mid \exists w' \in \Sigma^*. w' \odot w \in L\} .$$

- La fermeture par facteur de L , notée $\text{Fact}(L)$, est le langage formé par l'ensemble des facteurs des mots de L , défini comme :

$$\text{Fact}(L) = \{w \in \Sigma^* \mid \exists w', w'' \in \Sigma^*. w' \odot w \odot w'' \in L\} .$$

- La fermeture par extension de L , notée $\text{Ext}(L)$, est le langage formé par l'ensemble des extensions des mots de L , défini comme :

$$\text{Ext}(L) = \{w \in \Sigma^* \mid \exists w' \in L. w' \preceq w\} .$$

Définition 61 (Langage fermé par préfixe, suffixe et extension) Le langage L est dit :

- fermé par préfixe (ou préfixe-clos) si $L = \text{Pref}(L)$;
- fermé par suffixe (ou suffixe-clos) si $L = \text{Suf}(L)$;
- fermé par extension (ou extension-clos) si $L = \text{Ext}(L)$.

Définition 62 (Fermeture de Kleene) La Fermeture de Kleene de L , encore appelée étoile de Kleene de L , notée L^* , est l'ensemble défini inductivement par les deux règles suivantes :

- $\epsilon \in L^*$, et
- si $u \in L, v \in L^*$, alors $u \odot v \in L^*$,
- (de manière équivalente à la précédente règle : si $u \in L^*, v \in L$, alors $u \odot v \in L^*$).

Remarque 5 La fermeture de Kleene de L est l'ensemble des mots formés par un nombre fini de concaténations de mots de L :

$$L^* = \{\epsilon\} \cup \{u_0 \odot \dots \odot u_n \mid n \in \mathbb{N} \wedge \forall i. i \leq n \implies u_i \in L\} .$$

2.3 Exercices

2.3.1 Mots et concaténation

Exercice 1 (♠) — Concaténation de mots définis sous forme d'applications

Considérons l'alphabet $\{0, 1\}$. Nous considérons la définition (non inductive) de mot sous forme d'application (définition 52).

- La concaténation des mots 01 et 10 est le mot 0110.
 - La concaténation du mot vide ϵ et du mot 101 est le mot 101.
1. Donner la définition formelle des applications correspondant à ces 5 mots et montrer que les deux affirmations ci-dessus sont cohérentes avec la définition de l'opération de concaténation associée (définition 57).

Exercice 2 (♠♠) — Longueur et nombre d'occurrences d'un symbole

Considérons un alphabet Σ .

1. En utilisant la définition (non inductive) de mot comme une application (définition 52), définir la fonction qui donne la longueur d'un mot.
2. En utilisant la définition (non inductive) de mot comme une application, définir la fonction qui donne le nombre d'occurrences d'un symbole dans un mot.
3. Mêmes questions que précédemment en utilisant les deux définitions inductives de mot (définition 53 et définition 54).

Exercice 3 (♠♠) — Élément neutre de la concaténation

Considérons un alphabet Σ , ϵ_Σ le mot vide sur Σ et la définition (non inductive) de mot sous forme d'application (définition 52).

1. Démontrer que le mot ϵ_Σ est l'élément neutre de la concaténation, c'est-à-dire $\forall u \in \Sigma^*, u \odot \epsilon_\Sigma = \epsilon_\Sigma \odot u = u$.

Exercice 4 (♠♠♠) — Associativité de la concaténation

Considérons un alphabet Σ , et la définition non inductive de mot sous forme d'application (définition 52).

1. Montrer que l'opération de concaténation est associative, c'est-à-dire pour tout mot $u, u', u'' : (u \odot u') \odot u'' = u \odot (u' \odot u'')$.

Exercice 5 (♠♠♠) — Variantes de la concaténation, définition inductive

Nous considérons les définitions inductives des mots (définition 53 et définition 54). Nous souhaitons définir l'opération de concaténation $\odot : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ sur un alphabet Σ , de manière inductive et en utilisant l'une des définitions inductives des mots. L'opération de concaténation prend deux mots u et u' en paramètre et retourne un mot dont les symboles sont constitués des symboles de u dans l'ordre, suivis des symboles de u' dans l'ordre. La concaténation de deux mots u et u' de Σ^* reste notée $u \odot u'$.

1. Donner une définition inductive de l'opération de concaténation \odot , par induction sur le premier opérande.
2. Démontrer que le mot ϵ_{Σ} est élément neutre de l'opération de concaténation.
3. Donner une définition inductive de l'opération de concaténation \odot , par induction sur le second opérande.
4. Démontrer que l'opération de concaténation est associative, pour les deux définitions précédentes.

Remarque 6 (Simplification des notations) *En conséquence du résultat de l'exercice 5 d'une part et en observant que l'ensemble des mots de longueur 1 sur un alphabet est en bijection avec cet alphabet d'autre part, il nous est possible de confondre les deux opérateurs \cdot et \odot utilisés jusqu'à présent. Ainsi, dans la suite, nous utiliserons l'opérateur \cdot indifféremment comme opérateur de concaténation entre mots et symboles.*

Exercice 6 (♠♦♣) — Équivalence des définitions de mots

Considérons un alphabet Σ . Soit $MOTS$ l'ensemble des mots que l'on peut définir avec la définition de mot sous forme d'application (définition 52). Soit $MOTS_D$ l'ensemble des mots sur Σ que l'on peut définir avec la définition inductive des mots à droite (définition 53). Soit $MOTS_G$ l'ensemble des mots que l'on peut définir avec une définition inductive des mots à gauche (définition 54).

En supposant l'opérateur de concaténation associatif, et que ϵ_{Σ} est élément neutre de la concaténation des mots sur Σ , nous souhaitons démontrer que ces définitions sont équivalentes, c'est-à-dire que les ensembles de mots ainsi définis sont les mêmes.

1. Démontrer que $MOTS_D = MOTS_G$.
2. Démontrer que les trois ensembles $MOTS$, $MOTS_D$ et $MOTS_G$ sont égaux.

2.3.2 Langages

Exercice 7 (♠♦♣) — Puissance d'un alphabet

Rappelons que pour un alphabet Σ , Σ^k et $\Sigma^{\leq k}$ sont les ensembles des mots sur Σ respectivement de longueur égale à k et de longueur inférieure ou égale à k .

1. Donner le cardinal de Σ^k en fonction du cardinal de Σ .
2. Démontrer que $\forall k \in \mathbb{N}, \Sigma^{k+1} = \Sigma^k \cdot \Sigma^1$.
3. Démontrer que $\forall k \in \mathbb{N}, \Sigma^{k+1} = \Sigma^1 \cdot \Sigma^k$.
4. Démontrer le résultat sur la cardinalité de Σ^k .

Exercice 8 (♠♦♣) — Condition nécessaire et suffisante

Considérons Σ un alphabet, s un symbole de Σ et la proposition $\{s\} \cdot \Sigma^* = \Sigma^*$.

1. Montrer que cette proposition est toujours fausse.
2. Donner une condition nécessaire à la proposition.

3. Donner une condition *suffisante* à la proposition.
4. Donner, si cela est possible, une condition *nécessaire et suffisante* à la proposition.

2.3.3 Concaténation de langages

Exercice 9 (♠) — Concaténation avec le langage vide

Considérons un alphabet Σ et L un langage sur Σ . Nous nous intéressons à la concaténation de L avec le langage vide (noté \emptyset).

1. En utilisant la définition de la concaténation de langages (définition 59), démontrer que $L \cdot \emptyset = \emptyset \cdot L = \emptyset$.

Exercice 10 (♠♠♣) — Condition nécessaire et suffisante

Considérons Σ un alphabet et L un langage sur Σ . Considérons la proposition $L \cdot \Sigma^* = \Sigma^* \cdot L = L$.

1. Donner une condition *suffisante* à la proposition.
2. Donner une condition *nécessaire* à la proposition.
3. Donner, si cela est possible, une condition nécessaire et suffisante à la proposition.

Exercice 11 (♠♠♣) — Condition nécessaire et suffisante

Considérons Σ un alphabet et L un langage sur Σ . Considérons la proposition $L \cdot \Sigma^* \neq \Sigma^*$.

1. Donner une condition *nécessaire* à la proposition.
2. Donner une condition *suffisante* à la proposition.
3. Donner, si cela est possible, une condition nécessaire et suffisante à la proposition.

Exercice 12 (♠) — Concaténation de langages

La définition de concaténation de langage est dédiée à un alphabet Σ donné (définition 59). Nous nous intéressons à généraliser un peu cette définition. Considérons deux alphabets Σ_1 et Σ_2 .

1. Définir un opérateur de concaténation de langage prenant en paramètre un langage sur Σ_1 et un langage sur Σ_2 .

Exercice 13 (♠♠♣) — Cardinal et concaténation

Considérons un alphabet Σ . Nous nous intéressons à la concaténation de deux langages L_1 et L_2 définis sur Σ , notée $L_1 \cdot L_2$; voir définition 59.

1. Donner deux langages L_1 et L_2 tels que $L_1 \neq \emptyset$, $L_2 \neq \emptyset$ et $|L_1 \cdot L_2| < |L_1| \times |L_2|$.
2. Démontrer que $\forall L_1, L_2 \subseteq \Sigma^*, |L_1 \cdot L_2| \leq |L_1| \times |L_2|$.
3. Donner des conditions suffisantes pour que $|L_1 \cdot L_2| = |L_1| \times |L_2|$.

2.4 Indications pour résoudre les exercices

Indications pour l'exercice 1 (p. 32)

1. En utilisant la définition de mot sous forme d'application, définir les fonctions pour chacun des mots en donnant leur domaine, codomaine et la valeur en chaque position. Utiliser ensuite la définition de l'opérateur de concaténation.

Indications pour l'exercice 2 (p. 32)

1. Pour les domaines des fonctions à définir, considérer un intervalle d'entiers naturels commençant à 1. Voir le mot comme une fonction et utiliser son domaine.
2. Fonction à deux paramètres. Voir le mot comme une fonction et considérer le nombre d'éléments dans le domaine associé au symbole passé en paramètre.
3. Pour la longueur d'un mot, considérer les deux cas de la définition inductive et donner une valeur de la fonction pour chacun de ces cas. Dans le cas inductif, la valeur de la fonction est donnée en fonction de sa valeur sur un autre argument. Pour le nombre d'occurrences d'un symbole dans un mot, suivre le principe utilisé à la précédente question.

Indications pour l'exercice 3 (p. 32)

1. Considérer un mot $u \in \Sigma^*$ et comparer les fonctions $u \odot \epsilon_\Sigma$ et u .

Indications pour l'exercice 4 (p. 32)

1. Considérer trois mots quelconques u, u' et u'' et comparer de manière systématique les fonctions $u \odot (u' \odot u'')$ et $(u \odot u') \odot u''$.

Indications pour l'exercice 5 (p. 32)

1. Considérer un mot u quelconque. Distinguer les deux cas de la définition inductive.
 - Définir $\epsilon \odot u$.
 - Définir $(s \cdot u') \odot u$ pour un mot u' .
2. Faire une induction sur u .
3. Même principe que pour la première question.
4. Faire une induction sur le premier opérande dans la concaténation de u_1, u_2, u_3 .

Indications pour l'exercice 6 (p. 33)

1. Montrer l'inclusion des deux ensembles l'un dans l'autre. Pour chaque inclusion, procéder par induction.
2. Montrer l'inclusion des deux ensembles l'un dans l'autre.

- Pour montrer que $MOTS \subseteq MOTS_D$, considérer un élément de $MOTS$ et montrer qu'il peut être obtenu en appliquant les règles de construction de $MOTS_D$.
- Pour montrer que $MOTS_D \subseteq MOTS$, considérer un élément de $MOTS_D$ et l'exprimer comme un élément de $MOTS$ (une application) en fonction du nombre d'applications de la règle de construction inductive nécessaire pour obtenir cet élément.

Indications pour l'exercice 7 (p. 33)

1. Utiliser des exemples pour deviner le cardinal de Σ^k . Par exemple, pour $\Sigma = \{a, b\}$, il y a 4 mots de longueur 2 ($|\Sigma^2| = 4$) : $a \cdot a, a \cdot b, b \cdot a, b \cdot b$.
2. Utiliser les définitions de concaténation de mots (définition 57) et de langages (définition 59). Montrer l'inclusion des ensembles l'un dans l'autre.
3. Même principe que pour la preuve de la question précédente.
4. Faire une preuve par récurrence sur $k \in \mathbb{N}$.

Indications pour l'exercice 8 (p. 33)

1. Observer d'abord que $\{s\} \cdot \Sigma^*$ est la concaténation de deux langages :
 - le langage $\{s\}$ constitué d'un mot de longueur 1 et
 - le langage universel Σ^* .
 Ensuite, déterminer pourquoi les deux ensembles de part et d'autre de l'égalité ne peuvent jamais être égaux.
2. Considérer une proposition p et écrire le fait que p soit une condition nécessaire sous forme de proposition logique.
3. Considérer une proposition p et écrire le fait que p soit une condition suffisante sous forme de proposition logique.
4. Combiner les deux propositions logiques des deux questions précédentes.

Indications pour l'exercice 9 (p. 34)

1. Appliquer la définition de l'opérateur de concaténation de langages.

Indications pour l'exercice 10 (p. 34)

1. Déterminer des propriétés ou conditions sur L qui impliquent la propriété de l'énoncé.
2. Les mots du langage L peuvent se décomposer en trois parties.
3. Déterminer une condition commune solution des deux questions précédentes.

Indications pour l'exercice 11 (p. 34)

1. Considérer le mot vide.

Indications pour l'exercice 12 (p. 34)

1. Généraliser la définition de concaténation de langages en modifiant la signature de l'opérateur.

Indications pour l'exercice 13 (p. 34)

1. L'un des langages doit contenir ϵ .
2. Plusieurs démonstrations sont possibles. Par exemple, en déterminant le nombre de choix possibles lors de la formation d'un mot de $L_1 \cdot L_2$.

2.5 Solutions des exercices

2.5.1 Mots et concaténation

Solution de l'exercice 1 (page 32)

1. Dans cet exercice, les mots sont notés en orange, les indices des lettres sont notés en bleu et les symboles de l'alphabet sont notés en violet. Nous omettons les symboles de concaténation pour la lisibilité.
 - Le mot **01** est l'application $01 : \{1, 2\} \rightarrow \{0, 1\}$, telle que $01(1) = 0$, $01(2) = 1$ et $01(i)$ n'est pas défini pour $i > 2$.
 - Le mot **10** est l'application $10 : \{1, 2\} \rightarrow \{0, 1\}$, telle que $10(1) = 1$, $10(2) = 0$ et $10(i)$ n'est pas défini pour $i > 2$.
 - Le mot **0110** est l'application $0110 : \{1, 2, 3, 4\} \rightarrow \{0, 1\}$, telle que $0110(1) = 0$, $0110(2) = 1$, $0110(3) = 1$, $0110(4) = 0$ et $0110(i)$ n'est pas défini pour $i > 4$.
 - Le mot **ϵ** est l'application $\epsilon : \emptyset \rightarrow \{0, 1\}$ telle que $\epsilon(i)$ n'est pas défini pour i dans \mathbb{N} .
 - Le mot **101** est l'application $101 : \{1, 2, 3\} \rightarrow \{0, 1\}$, telle que $101(1) = 1$, $101(2) = 0$, $101(3) = 1$ et $101(i)$ n'est pas défini pour $i > 3$.

Les deux concaténations définies dans la question sont cohérentes avec la définition de concaténation car on peut observer que $|\{1, 2\}| = 2$, $|\{1, 2, 3, 4\}| = 4$ et $|\emptyset| = 0$.

Solution de l'exercice 2 (page 32)

Dans les réponses suivantes, E est un sous-ensemble de \mathbb{N}^* sans interruption (c'est-à-dire un intervalle) commençant à 1.

1. Nous définissons l'application longueur : $(E \rightarrow \Sigma) \rightarrow \mathbb{N}$ qui prend en paramètre un mot sur Σ et retourne un entier naturel, et telle que pour tout mot u de $E \rightarrow \Sigma$: $\text{longueur}(u) = |\text{domaine}(u)|$.
2. Nous définissons la fonction occurrences : $(E \rightarrow \Sigma) \times \Sigma \rightarrow \mathbb{N}$ qui prend en paramètre un mot u sur Σ et un symbole de Σ , et telle que pour tout mot u de $E \rightarrow \Sigma$ et symbole s de Σ : $\text{occurrences}(u, s) = |\{e \in E \mid u(e) = s\}|$.

3. Notons $MOTS_{\Sigma}$ l'ensemble des mots obtenus selon la définition inductive à droite sur l'alphabet Σ .

La fonction longueur : $MOTS_{\Sigma} \rightarrow \mathbb{N}$ est définie par :

- longueur(ϵ) = 0,
- longueur($u \cdot s$) = 1 + longueur(u), pour $u \in MOTS_{\Sigma}$ et $s \in \Sigma$.

La fonction occurrences : $MOTS_{\Sigma} \times \Sigma \rightarrow \mathbb{N}$ est définie par :

- occurrences(ϵ, s) = 0, pour tout $s \in \Sigma$,
- pour $u \in MOTS_{\Sigma}$ et $s, s' \in \Sigma$

$$\text{occurrences}(u \cdot s, s') = \begin{cases} 1 + \text{occurrences}(u) & \text{si } s = s', \\ \text{occurrences}(u) & \text{sinon.} \end{cases}$$

Les définitions de ces fonctions avec la définition inductive à gauche sont similaires (elles sont obtenues en « inversant » u et s dans occurrences($u \cdot s, s'$)).

Solution de l'exercice 3 (page 32)

1. Soit $n \in \mathbb{N}$ et $u \in \Sigma^*$ un mot de longueur n . Le mot u est l'application $u : \{1, \dots, n\} \rightarrow \Sigma$; en particulier $u(i)$ n'est pas défini pour $i > |u|$. Le mot vide ϵ_{Σ} est l'application $\epsilon_{\Sigma} : \emptyset \rightarrow \Sigma$ et $|\epsilon_{\Sigma}| = 0$. D'après la définition de l'opérateur de concaténation, $u \odot \epsilon_{\Sigma}$ est l'application $u \odot \epsilon_{\Sigma} : [1, |u| + |\epsilon|] \rightarrow \Sigma$ telle que $(u \odot \epsilon_{\Sigma})(i) = u(i)$ pour $i \leq |u|$ et $(u \odot \epsilon_{\Sigma})(i)$ n'est pas défini pour $i > |u|$. Ainsi, les définitions de u et $u \odot \epsilon_{\Sigma}$ sont les mêmes.

Le même raisonnement peut être suivi pour montrer que $\epsilon_{\Sigma} \odot u = u$.

Solution de l'exercice 4 (page 32)

1. Nous devons montrer que :

$$\forall u, u', u'' \in \Sigma^*. u \odot (u' \odot u'') = (u \odot u') \odot u''$$

Soient $u, u', u'' \in \Sigma^*$. Calculons $u \odot (u' \odot u'')$. Le mot $u' \odot u''$ est l'application définie par :

$$\forall i \in [1, |u'| + |u''|]. u' \odot u''(i) = \begin{cases} u'(i) & \text{si } i \in [1, |u'|], \\ u''(i - |u'|) & \text{si } i \in [|u'| + 1, |u'| + |u''|]. \end{cases}$$

Comme $u' \odot u''$ est de longueur $|u'| + |u''|$, nous en déduisons que le mot $u \odot (u' \odot u'')$ est l'application définie par :

$$\forall i \in [1, |u| + (|u'| + |u''|)]. u \odot (u' \odot u'')(i) = \begin{cases} u(i) & \text{si } i \in [1, |u|], \\ (u' \odot u'')(i - |u|) & \text{si } i \in [|u| + 1, |u| + (|u'| + |u''|)]. \end{cases}$$

D'après la définition de $u' \odot u''$ ci-dessus, nous en déduisons que le mot $u \odot (u' \odot u'')$ est l'application définie par :

$$\forall i \in [1, |u| + (|u'| + |u''|)]. u \odot (u' \odot u'')(i) = \begin{cases} u(i) & \text{si } i \in [1, |u|], \\ u'(i - |u|) & \text{si } i \in [|u| + 1, |u| + |u'|], \\ u''(i - |u| - |u'|) & \text{si } i \in [|u| + |u'| + 1, |u| + |u'| + |u''|]. \end{cases}$$

De manière similaire, calculons $(u \odot u') \odot u''$. Le mot $u \odot u'$ est l'application définie par :

$$\forall i \in [1, |u| + |u'|]. u \odot u'(i) = \begin{cases} u(i) & \text{si } i \in [1, |u|], \\ u'(i - |u|) & \text{si } i \in [|u| + 1, |u| + |u'|]. \end{cases}$$

Nous en déduisons, comme $u \odot u'$ est de longueur $|u| + |u'|$, que le mot $(u \odot u') \odot u''$ est l'application définie par :

$$\forall i \in [1, (|u| + |u'|) + |u''|]. (u \odot u') \odot u''(i) = \begin{cases} (u \odot u')(i) & \text{si } i \in [1, |u| + |u'|], \\ u''(i - (|u| + |u'|)) & \text{si } i \in [|u| + |u'| + 1, (|u| + |u'|) + |u''|]. \end{cases}$$

D'après la définition de $u \odot u'$ ci-dessus, nous en déduisons que le mot $(u \odot u') \odot u''$ est l'application définie par :

$$\forall i \in [1, (|u| + |u'|) + |u''|]. u \odot (u' \odot u'')(i) = \begin{cases} u(i) & \text{si } i \in [1, |u|], \\ u'(i - |u|) & \text{si } i \in [|u| + 1, |u| + |u'|], \\ u''(i - |u| - |u'|) & \text{si } i \in [|u| + |u'| + 1, |u| + |u'| + |u''|]. \end{cases}$$

Solution de l'exercice 5 (page 32)

1. Nous définissons l'opérateur \odot par :

- $\epsilon \odot u = u$, pour $u \in \Sigma^*$,
- $(s \cdot u') \odot u = s \cdot (u' \odot u)$, pour $u, u' \in \Sigma^*$ et $s \in \Sigma$ – notons que l'opérateur \cdot est celui apparaissant dans la définition inductive des mots.

2. Nous devons démontrer que $\forall u \in \Sigma^*. u \odot \epsilon_\Sigma = \epsilon_\Sigma \odot u = u$; ce que nous démontrons par induction sur u . Observons que nous avons déjà $\epsilon_\Sigma \odot u = u$, pour tout mot $u \in \Sigma^*$ par définition de l'opérateur de concaténation.

- Cas de base. Considérons le cas où $u = \epsilon_\Sigma$. Par définition de l'opérateur de concaténation donné dans la précédente question, $\epsilon_\Sigma \odot \epsilon_\Sigma = \epsilon_\Sigma$.
- Pas d'induction. Soit $u \in \Sigma^*$, supposons que $u \odot \epsilon_\Sigma = \epsilon_\Sigma \odot u = u$. Considérons un symbole $s \in \Sigma$ et le mot $s \cdot u$, montrons que $(s \cdot u) \odot \epsilon_\Sigma = \epsilon_\Sigma \odot (s \cdot u) = s \cdot u$. Par définition de l'opérateur de concaténation, $(s \cdot u) \odot \epsilon_\Sigma = s \cdot (u \odot \epsilon_\Sigma)$. En utilisant l'hypothèse d'induction, nous obtenons $s \cdot (u \odot \epsilon_\Sigma) = s \cdot u$.

3. Nous définissons l'opérateur \odot' par :

- $u \odot' \epsilon = u$, pour $u \in \Sigma^*$,
- $u \odot' (u' \cdot s) = (u \odot' u') \cdot s$, pour $u, u' \in \Sigma^*$ et $s \in \Sigma$.

4. Considérons la première définition. Nous devons montrer que $\forall u_1, u_2, u_3 \in \Sigma^*. u_1 \odot (u_2 \odot u_3) = (u_1 \odot u_2) \odot u_3$. Nous faisons une preuve par induction sur le premier opérande.

- Considérons le cas où $u_1 = \epsilon$. Soient u_2, u_3 deux mots. D'une part, nous avons $\epsilon \odot (u_2 \odot u_3) = u_2 \odot u_3$ par définition de l'opérateur de concaténation. D'autre part, nous avons également $(\epsilon \odot u_2) \odot u_3 = u_2 \odot u_3$.

- Soient u_2, u_3 deux mots. Supposons que $u_1 \odot (u_2 \odot u_3) = (u_1 \odot u_2) \odot u_3$ pour un certain mot $u_1 \in \Sigma^*$. Considérons le mot $u_1 = s \cdot u$ pour un certain symbole $s \in \Sigma$. Nous avons $(s \odot u_1) \odot (u_2 \odot u_3) = s \cdot (u_1 \odot (u_2 \odot u_3))$ par définition de \odot . D'après l'hypothèse d'induction, nous avons $s \cdot (u_1 \odot (u_2 \odot u_3)) = s \cdot ((u_1 \odot u_2) \odot u_3)$. D'autre part, nous avons $s \cdot ((u_1 \odot u_2) \odot u_3) = (s \cdot (u_1 \odot u_2)) \odot u_3 = ((s \cdot u_1) \odot u_2) \odot u_3$, par définition de \odot .

La preuve pour le deuxième opérateur se réalise de manière similaire.

Solution de l'exercice 6 (page 33)

1. Nous montrons l'inclusion des deux ensembles, l'un dans l'autre.

Montrons que $MOTS_D \subseteq MOTS_G$ en montrant que tout mot de $MOTS_D$ est un mot de $MOTS_G$, par induction sur les mots de $MOTS_D$.

- Cas de base. Considérons le mot $\epsilon \in MOTS_D$, nous avons $\epsilon \in MOTS_G$, par définition de $MOTS_G$.
- Pas d'induction. Supposons la propriété vérifiée pour un mot u de $MOTS_D$. Soit s un symbole de Σ , considérons le mot $u \cdot s \in MOTS_D$. En utilisant l'hypothèse d'induction sur le mot u , le mot $u \cdot s$ peut s'écrire $(s' \cdot u') \cdot s$, avec $u' \in MOTS_G$. En utilisant l'associativité de la concaténation, $u \cdot s$ peut s'écrire $s' \cdot (u' \cdot s)$. Le symbole s est également un mot de $MOTS_G$ de longueur 1 (il est obtenu en concaténant s avec ϵ). Ainsi, $u' \cdot s$ est la concaténation de deux mots de $MOTS_G$ et est donc un mot de $MOTS_G$. (On peut montrer aisément par induction sur le premier argument de la concaténation que la concaténation de deux mots de $MOTS_G$ est un mot de $MOTS_G$.) En utilisant la définition de $MOTS_G$, $s' \cdot (u' \cdot s)$ est la concaténation de $s' \in \Sigma$ et un mot de $MOTS_G$, c'est donc un mot de $MOTS_G$.

Montrons que $MOTS_G \subseteq MOTS_D$ en montrant que tout mot de $MOTS_G$ est un mot de $MOTS_D$, par induction sur les mots de $MOTS_G$.

- Cas de base. Considérons le mot $\epsilon \in MOTS_G$, nous avons $\epsilon \in MOTS_D$.
- Pas d'induction. Supposons la propriété vérifiée pour un mot u de $MOTS_G$. Soit s un symbole de Σ , considérons le mot $s \cdot u \in MOTS_G$. En utilisant l'hypothèse d'induction sur le mot u , le mot $s \cdot u$ peut s'écrire $s \cdot (u' \cdot s')$. En utilisant l'associativité de la concaténation, $u \cdot s$ peut s'écrire $(s \cdot u') \cdot s'$. Le symbole s est également un mot de $MOTS_D$ de longueur 1 (il est obtenu en concaténant s avec ϵ). Ainsi, $s \cdot u'$ est la concaténation de deux mots de $MOTS_D$ et est donc un mot de $MOTS_D$. (On peut montrer aisément par induction sur le deuxième argument de la concaténation que la concaténation de deux mots de $MOTS_D$ est un mot de $MOTS_D$.) En utilisant la définition de $MOTS_D$, $(s \cdot u') \cdot s'$ est la concaténation de $s' \in \Sigma$ et un mot de $MOTS_D$, c'est donc un mot de $MOTS_D$.

2. Nous montrons l'égalité entre $MOTS$ et $MOTS_D$, c'est-à-dire l'inclusion des deux ensembles l'un dans l'autre ; ce qui donne le résultat attendu en utilisant le résultat de la question précédente.

- Preuve que $MOTS \subseteq MOTS_D$. Soit $u : E \rightarrow \Sigma$ un mot de $MOTS$, nous montrons que $u \in MOTS_D$ c'est-à-dire qu'il est obtenu par application de

la règle produisant, pour un mot $u \in MOTS_D$ et un symbole a de Σ , le mot $u \cdot a \in MOTS$, ceci à partir de l'élément de base ϵ . Si le mot u est le mot ϵ , alors le résultat est immédiat. Sinon, le mot u est obtenu en réalisant $|u|$ applications de la règle de construction des mots à partir du mot ϵ . La $i^{\text{ème}}$ application de la règle permettant de construire le préfixe de longueur i , à partir du préfixe de longueur $i - 1$ et du symbole $u(i)$. Ceci pouvant être démontré facilement par récurrence sur i . Ainsi, $|u|$ applications de la règle de construction permet d'obtenir le préfixe de longueur $|u|$ de u , c'est-à-dire u lui-même.

- Preuve que $MOTS_D \subseteq MOTS$. Soit $u \in MOTS_D$, nous montrons que $u \in MOTS$. Soit n le nombre d'applications de la règle inductive et soit a_i le symbole de Σ utilisé pour la $i^{\text{ème}}$ application de la règle. Considérons l'application $u' : [1, n] \rightarrow \Sigma$ telle que $u'(i) = a_i$. Alors, on montre par induction sur i que pour $i \geq 1$, les préfixes de longueur i de u et u' sont les mêmes.

2.5.2 Langages

Solution de l'exercice 7 (page 33)

1. Nous avons : $|\Sigma^k| = |\Sigma|^k$. En effet, $\Sigma^k = \{s_1 \cdots s_k \mid s_1 \in \Sigma \wedge \dots \wedge s_k \in \Sigma\}$ et il y a donc $|\Sigma|$ choix possibles pour chaque s_i . Donc $|\Sigma^k| = \underbrace{|\Sigma| \times \dots \times |\Sigma|}_{k \text{ fois}} = |\Sigma|^k$.
2. La preuve découle directement des définitions de concaténation de mots (définition 57) et de langages (définition 59). Nous prouvons l'inclusion des ensembles l'un dans l'autre, dans les deux sens.
 - Soit $k \in \mathbb{N}$. Soit $u \in \Sigma^{k+1}$, u est un mot de longueur $k + 1$, c'est-à-dire une application de $[1, k + 1]$ vers Σ . En utilisant la définition de concaténation de mot, nous en déduisons l'existence de deux mots $v : [1, k] \rightarrow \Sigma$ et $v' : \{1\} \rightarrow \Sigma$ tels que $\forall i \in [1, k]. v(i) = u(i)$ et $v'(1) = u(k)$; c'est-à-dire $u = v \cdot v'$. De plus, $v \in \Sigma^k$ et $v' \in \Sigma^1$ car v et v' sont respectivement des mots de longueurs k et 1. Nous en déduisons que pour tout $u \in \Sigma$, il existe $v \in \Sigma^k$ et $v' \in \Sigma^1$ tels que $u = v \cdot v'$.
 - Soit $u \cdot v \in \Sigma^k \cdot \Sigma^1$, nous pouvons suivre le raisonnement précédent en sens inverse pour aboutir à $u \cdot v \in \Sigma^{k+1}$. Donc $\Sigma^{k+1} = \Sigma^1 \cdot \Sigma^k$.
3. Le principe de la preuve est le même que celui de la preuve de la réponse précédente.
4. Nous faisons une preuve par récurrence sur k .
 - Cas de base. Pour $k = 0$, $|\Sigma^0| = 1$, il y a un seul mot (ϵ) de longueur 0.
 - Pas d'induction. Supposons la propriété vérifiée pour une certaine $k \in \mathbb{N}$. Considérons Σ^{k+1} . D'après les questions précédentes, $\Sigma^{k+1} = \Sigma^k \cdot \Sigma$. Ainsi, d'après la définition de l'opérateur de concaténation des langages, nous avons $\Sigma^{k+1} = \{u_k \cdot u_1 \mid u_k \in \Sigma^k \wedge u_1 \in \Sigma^1\}$. Pour former un mot de Σ^{k+1} , on choisit un mot dans Σ^k et un mot dans Σ^1 . Ainsi, pour former un mot de Σ^{k+1} , il y a $|\Sigma^k| \times |\Sigma^1|$ possibilités toutes formant un mot unique. De plus, comme $|\Sigma^k| = |\Sigma|^k$ (hypothèse de récurrence) et $|\Sigma^1| = |\Sigma|$, nous obtenons le résultat attendu.

Solution de l'exercice 8 (page 33)

Rappelons que, étant donné une proposition logique p :

- une condition nécessaire pour p est une proposition logique p_n telle que la proposition logique $p \implies p_n$ soit vraie,
- une condition suffisante pour p est une proposition logique p_s telle que la proposition logique $p_s \implies p$ soit vraie.

Rappelons qu'une implication $p \implies q$ est logiquement équivalente à $\neg p \vee q$.

1. Puisque $s \in \Sigma$, $\Sigma \neq \emptyset$ et donc $\Sigma^* \neq \emptyset$. Ainsi, la concaténation des langages $\{s\}$ et Σ^* , $\{s\} \cdot \Sigma^*$, n'est pas vide non plus, et tous les mots de $\{s\} \cdot \Sigma^*$ sont préfixés par s , par définition de la concaténation des langages. Ainsi, $\epsilon \notin \{s\} \cdot \Sigma^*$ puisque s n'est pas un préfixe de ϵ . Or, par définition du langage universel, $\epsilon \in \Sigma^*$. Comme $\epsilon \in \Sigma^*$ et $\epsilon \notin \{s\} \cdot \Sigma^*$, $\Sigma^* \neq \{s\} \cdot \Sigma^*$. La proposition $\{s\} \cdot \Sigma^* = \Sigma^*$ est donc logiquement équivalente à F .
2. La proposition $(\neg(\{s\} \cdot \Sigma^* = \Sigma^*)) \vee p_n$ est logiquement équivalente à $V \vee p_n$. Cette proposition est vraie pour toute proposition p_n . Donc, toute proposition est nécessaire à $\{s\} \cdot \Sigma^* = \Sigma^*$.
3. La proposition $(\neg p_s) \vee (\{s\} \cdot \Sigma^* = \Sigma^*)$ est logiquement équivalent à $(\neg p_s) \vee F$. Pour que cette dernière proposition soit vraie, il faut choisir $p_s = F$. Donc, seule la proposition F est suffisante à $\{s\} \cdot \Sigma^* = \Sigma^*$.
4. Seule la proposition F est nécessaire et suffisante.

2.5.3 Concaténation de langages**Solution de l'exercice 9 (page 34)**

1. Soit L un langage. Alors, d'après la définition de l'opérateur de concaténation de langages, $L \cdot \emptyset = \{u_L \cdot u_\emptyset \mid u_L \in L \wedge u_\emptyset \in \emptyset\}$. Or, $u_\emptyset \in \emptyset$ est impossible. La condition définissant l'ensemble est insatisfiable (elle est équivalente à la proposition toujours fausse). Donc $L \cdot \emptyset = \emptyset$.

Solution de l'exercice 10 (page 34)

1. Nous avons quatre conditions suffisantes.

- $\epsilon \in L$.
- $L = \emptyset$ est une condition suffisante car si $L = \emptyset$ alors $L \cdot \Sigma^* = \emptyset$ et $\Sigma^* \cdot L = \emptyset$.
- $L = \Sigma^*$ est une condition suffisante car si $L = \Sigma^*$ alors $L \cdot \Sigma^* = \Sigma^*$ et $\Sigma^* \cdot L = \Sigma^*$.
- $\exists L_0 \subseteq \Sigma^*. L = \Sigma^* \cdot L_0 \cdot \Sigma^*$ est une condition suffisante. En effet, nous avons :

$$\begin{aligned} L \cdot \Sigma^* &= (\Sigma^* \cdot L_0 \cdot \Sigma^*) \cdot \Sigma^* \\ &= \Sigma^* \cdot L_0 \cdot \Sigma^* \cdot \Sigma^* \\ &= \Sigma^* \cdot L_0 \cdot \Sigma^* \\ &= L \end{aligned}$$

De manière similaire, nous pouvons montrer que $\Sigma^* \cdot L = L$.

2. Une condition nécessaire est $\exists L_0 \subseteq \Sigma^*, L = \Sigma^* \cdot L_0 \cdot \Sigma^*$. Montrons que $(\Sigma^* \cdot L = L \cdot \Sigma^* = L) \implies (\exists L_0 \subseteq \Sigma^*, L = \Sigma^* \cdot L_0 \cdot \Sigma^*)$, en montrant la contraposée. Supposons que $(\Sigma^* \cdot L = L \cdot \Sigma^* = L)$. Nous avons $(L \cdot \Sigma^*) \cdot \Sigma^* = L \cdot \Sigma^* = L$.
3. Nous avons montré que $\exists L_0 \subseteq \Sigma^*, L = \Sigma^* \cdot L_0 \cdot \Sigma^*$ est une condition nécessaire et suffisante.

Solution de l'exercice 11 (page 34)

1. On a $L \cdot \Sigma^* \neq \Sigma^* \implies \epsilon \notin L$, ainsi $\epsilon \notin L$ est une condition nécessaire. Pour le montrer, nous considérons la contraposée $\epsilon \in L \implies L \cdot \Sigma^* = \Sigma^*$. Supposons que $\epsilon \in L$. L'inclusion $L \cdot \Sigma^* \subseteq \Sigma^*$ est immédiate. Soit $u \in \Sigma^*$, nous avons $u = \epsilon \cdot u$ et donc $u \in L \cdot \Sigma^*$ d'après la définition de concaténation de langage.
2. La condition $\epsilon \notin L$ est une condition suffisante car $\epsilon \notin L \implies L \cdot \Sigma^* \neq \Sigma^*$. Supposons que $\epsilon \notin L$. Tous les mots de L sont de longueur supérieure ou égale à 1. Ainsi, d'après la définition de concaténation de langage, tous les mots de $L \cdot \Sigma^*$ sont également de longueur supérieure ou égale à 1. Nous avons donc que $\epsilon \notin L \cdot \Sigma^*$ alors que $\epsilon \in L$.
3. En utilisant les résultats des deux questions précédentes, nous en déduisons que $\epsilon \notin L$ est une condition nécessaire et suffisante.

Solution de l'exercice 12 (page 34)

1. L'opérateur de concaténation de langages est noté \cdot et est défini comme suit.

$$\begin{array}{rcl} \cdot & : & \mathcal{P}(\Sigma_1^*) \times \mathcal{P}(\Sigma_2^*) \rightarrow \mathcal{P}(\Sigma_1 \cup \Sigma_2)^* \\ L_1 \cdot L_2 & = & \{u_1 \cdot u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\} \end{array}$$

(L_1 et L_2 sont deux langages sur des alphabets Σ_1 et Σ_2 , respectivement.)

Solution de l'exercice 13 (page 34)

1. Considérons les langages $L_1 = \{\epsilon, a\}$ et $L_2 = \{b, a \cdot b\}$. Nous avons $L_1 \cdot L_2 = \{b, a \cdot b, a \cdot a \cdot b\}$. Nous avons $|L_1 \cdot L_2| = 3$, $|L_1| = 2$ et $|L_2| = 2$.
2. Nous proposons trois démonstrations.
 - La première démonstration consiste d'abord à remarquer que, d'après la définition de la concaténation de langages, $L_1 \cdot L_2 = \{u_1 \cdot u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\}$. Ainsi, chaque mot $u_1 \cdot u_2$ dans $L_1 \cdot L_2$ est construit en choisissant un mot u_1 dans L_1 et un mot u_2 dans L_2 . Pour ces choix, nous avons $|L_1|$ et $|L_2|$ choix possibles, respectivement. Comme il peut avoir des choix qui mènent à des mots qui, une fois concaténés, produisent le même mot, cela nous donne l'inégalité annoncée.
 - La deuxième démonstration est une récurrence sur $|L_1|$.
 - Pour $|L_1| = 0$, nous avons nécessairement $L_1 = \emptyset$. Ainsi d'après la définition de l'opérateur de concaténation sur les langages, nous obtenons $L_1 \cdot L_2 = \emptyset$. Ainsi $|L_1 \cdot L_2| = 0$.
 - Supposons la propriété vraie pour les langages (utilisés comme premier opérande) de longueur inférieure ou égale n , pour un certain $n \in \mathbb{N}$. Considérons un langage L_1 tel que $|L_1| = n + 1$. L_1 peut s'écrire $L'_1 \cup \{w\}$ où w est un

mot tel que $w \notin L_1$. Nous avons $L_1 \cdot L_2 = L'_1 \cup \{w\} \cdot L_2 = \{u_1 \cdot u_2 \mid u_1 \in L'_1 \cup \{w\} \text{ et } u_2 \in L_2\}$. Considérons la condition $u_1 \in L'_1 \cup \{w\}$ et $u_2 \in L_2$. Cette condition est logiquement équivalente à $(u_1 \in L'_1 \text{ et } u_2 \in L_2)$ ou $w \in \{w\}$ et $u_2 \in L_2$. Ainsi $L_1 \times L_2 = L'_1 \cdot L_2 \cup \{w\} \cdot L_2$ et donc $|L_1 \cdot L_2| = |L'_1 \cdot L_2| + |\{w\} \cdot L_2| - |L'_1 \cdot L_2 \cap \{w\} \cdot L_2| \leq |L'_1 \cdot L_2| + |\{w\} \cdot L_2|$. En utilisant l'hypothèse de récurrence, nous avons $|L'_1 \cdot L_2| \leq |L'_1| \times |L_2|$ et $|\{w\} \cdot L_2| \leq |L_2|$ et obtenons ainsi l'inégalité recherchée.

- La troisième démonstration consiste à exhiber une injection de $L_1 \cdot L_2$ vers $L_1 \times L_2$. Soient (u_1, u_2) et (u'_1, u'_2) deux éléments de $L_1 \times L_2$ tels que $(u_1, u_2) = (u'_1, u'_2)$. D'après la définition de couple, ceci implique que $u_1 = u'_1$ et $u_2 = u'_2$. par définition de la concaténation, nous avons $u_1 \cdot u_2 = u'_1 \cdot u'_2$. Ainsi, il existe bien une injection de $L_1 \cdot L_2$ vers $L_1 \times L_2$ et donc $|L_1 \cdot L_2| \leq |L_1 \times L_2| = |L_1| \times |L_2|$.
3. Une première condition suffisante est que les alphabets sur lesquels sont définis les langages sont disjoints. Une seconde condition suffisante est d'éviter les « collisions » en formant des mots en prenant des mots des deux langages L_1 et L_2 . Formellement, $\forall u_1, v_1 \in L_1. \forall u_2, v_2 \in L_2. (u_1 \neq v_1 \vee u_2 \neq v_2 \implies u_1 \cdot u_2 \neq v_1 \cdot v_2)$.

Chapitre 3

Automates déterministes

3.1 Résumé intuitif du chapitre

Dans ce chapitre, nous introduisons la notion d'automate déterministe, un concept fondamental en informatique. Un automate peut être vu comme une machine simple lisant un mot (voir chapitre 2) et produisant un résultat. Le mécanisme de cette machine utilise des états pour mémoriser les symboles reçus, et peut ainsi réagir différemment selon différents mots. Les entrées des automates sont donc des symboles pris dans un alphabet auquel l'automate réagit. Lors de chaque réception de symbole, l'automate peut changer d'état et produit un nouveau résultat. Les automates que nous considérons dans ce livre produisent deux sorties pour indiquer si la séquence de symboles reçues est acceptée ou non.

Plus précisément, nous abordons dans ce chapitre les concepts d'**automate** (déterministe), d'**exécution** et de **langage reconnu**. Comme indiqué dans le paragraphe précédent, un automate sera décrit par un ensemble d'états, un alphabet de symboles auxquels il réagit ainsi qu'une relation (dite de transition) qui décrit les changements d'états en fonction des symboles reçus. Un automate sera doté d'un état initial : l'état dans lequel il se trouve avant la réception de symboles. Aussi, pour indiquer quelles sont les mots acceptées, certains états seront marqués comme étant **accepteurs** (on dit aussi **terminaux**) : lorsqu'un automate, après avoir lu un symbole, entre dans un état accepteur, cela signifie que la suite de symboles reçue jusqu'à présent est acceptée. Ceci implique également que toutes les séquences d'entrées qui mènent l'automate dans cet état seront acceptées de manière équivalente.

Les automates considérés dans ce chapitre sont dits **déterministes**. Le déterminisme se caractérise par le fait qu'à chaque réception de symbole, l'automate atteint un état unique, et n'a donc pas le choix de l'état à atteindre avec le symbole reçu. Ceci implique également que pour un mot donné, l'état atteint après lecture du mot complet est soit terminal soit non terminal et donc que la sortie de l'automate est unique. La notion d'exécution d'un automate décrit de manière précise, l'évolution d'un automate lorsque celui-ci lit un mot ; cette évolution est décrite par une séquence de **configurations** dictée par une **relation de dérivation** entre configurations. Le langage reconnu par l'automate sera simplement l'ensemble des mots (un langage) acceptés par l'automate, c'est-à-dire qui mènent l'automate dans un état terminal.

Nous introduisons également la **fondation de transition étendue aux mots** définissant l'état atteint à partir d'un autre état après lecture d'un mot. Ceci nous permet de redéfinir la condition d'acceptation d'un mot par un automate.

Nous introduisons finalement les notions d'**accessibilité** et de **coaccessibilité** d'un état. Un état est dit **accessible** lorsqu'on peut l'atteindre depuis l'état initial, en suivant la fonction de transition. Un état est dit **coaccessible** lorsqu'on peut atteindre un état terminal depuis cet état, en suivant la fonction de transition.

3.2 Les notions essentielles

Remarque 7 *Comme suite aux exercices du chapitre précédent, nous ne faisons plus la distinction entre l'opérateur de concaténation entre mots (noté \odot) et l'opérateur de concaténation entre un symbole et un mot (noté \cdot) et utilisons le symbole \cdot pour les deux.*

3.2.1 Définition et langage reconnu

Définition 63 (Automate à nombre fini d'états et déterministe) *Un automate à nombre fini d'états et déterministe (abrégé AFED) est donné par un quintuplet $(Q, \Sigma, q_{\text{init}}, \delta, F)$ tel que :*

- Q est un ensemble fini non vide dont les éléments sont appelés états ;
- Σ est l'alphabet de l'automate ;
- $q_{\text{init}} \in Q$ est l'état initial ;
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition de l'automate, elle peut être partielle (voir remarque 9) ;
- $F \subseteq Q$ est l'ensemble des états états accepteurs.¹

Définition 64 (Automate déterministe complet) *Un AFED est dit complet si sa fonction de transition est totale : δ est une application). C'est-à-dire, si tous les états ont une transition sur chaque symbole de Σ .*

Définition 65 (Représentations d'un automate) *Nous utilisons deux représentations des automates : graphique et tabulaire.*

- *Dans la représentation graphique, l'automate est représenté sous forme de graphe. Les états sont les nœuds (cercles) et les transitions des arrêtes entre ces nœuds (flèches reliant les cercles). L'état initial est l'état avec une arête ne provenant pas d'un autre état et les états accepteurs sont représentés par cercles doublés. Les symboles des transitions sont indiqués comme une étiquette. Une transition étiquetée par un ensemble de symboles représente l'ensemble des transitions où chaque transition est étiquetée par un symbole de cet ensemble.*
- *Dans la représentation tabulaire, l'automate est représenté sous forme de tableau. Les états sont en colonnes et les symboles en ligne (ou inversement). Dans une case du tableau, on retrouve l'état destination de la transition sur l'état et le symbole associés à la case. Nous utilisons une flèche pour indiquer l'état initial et des étoiles pour indiquer les états accepteurs.*

¹. Dans la littérature, les état accepteurs sont aussi appelés *états terminaux*. Nous utiliserons le terme *état accepteur*.

Deux représentations du même automate, sous formes graphique et tabulaire, sont illustrées dans la figure 3.1.

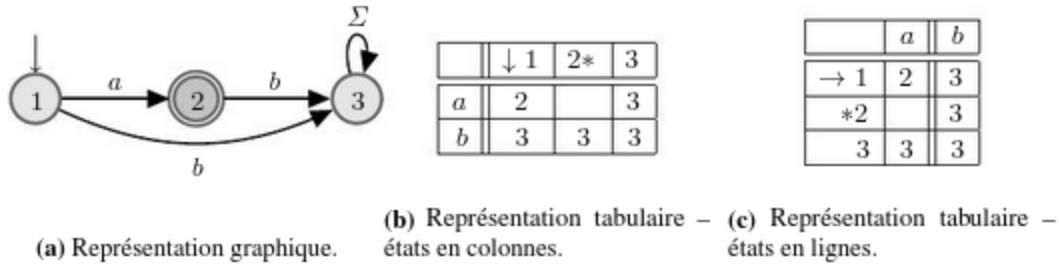


Figure 3.1 – Représentations d'un automate sous forme graphique et tabulaire.

Remarque 8 (Déterminisme) Le déterminisme de l'automate provient du fait que les transitions (et donc les changements d'états de l'automate) sont régis par une fonction : à un état et symbole donné, il y a au plus un état associé.

Remarque 9 La fonction de transition d'un automate est partielle, lorsqu'il existe au moins un état et un symbole pour lesquels la fonction de transition n'est pas définie. C'est par exemple le cas de l'automate représenté dans la figure 3.1 pour l'état 2 et le symbole a .

Dans la suite, nous utilisons un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

Définition 66 (Configuration d'un AFED) Une configuration de l'automate A sur un mot u est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.

Définition 67 (Relation de dérivation d'un AFED) La relation de dérivation entre configurations de A , notée \rightarrow , est définie comme suit :

$$\forall q \in Q. \forall a \in \Sigma. \forall u \in \Sigma^*. (q, a \cdot u) \rightarrow (q', u) \iff \delta(q, a) = q'.$$

La relation \rightarrow est donc un sous-ensemble de $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$.

Définition 68 (Exécution d'un AFED) L'exécution de A sur un mot u à partir d'un état $q_1 \in Q$ est la séquence de configurations $(q_1, u_1) \cdots (q_n, u_n)$ telle que : $u_1 = u$ et $\forall i \in [1, n-1]. (q_i, u_i) \rightarrow (q_{i+1}, u_{i+1})$.

Remarque 10 (À propos des configurations) L'exécution d'un mot se définit par rapport à un état qui n'est pas forcément l'état initial. À partir d'une configuration, une exécution se termine lorsqu'il n'est pas possible de trouver une configuration suivante avec la relation de dérivation. Une exécution peut donc se terminer dans une configuration (q, u) avec $u \neq \epsilon$, on parle de configuration bloquante. Une exécution peut également se terminer dans une configuration (q, ϵ) , on parle de configuration terminale. Pour une configuration terminale, on parle de configuration acceptante ou configuration non acceptante lorsque l'état de cette configuration est accepteur ou non accepteur, respectivement.

Définition 69 (Acceptation et non-acceptation d'un mot par un AFED) *Un mot $u \in \Sigma^*$ est accepté par A , si l'exécution de A sur u (à partir de son état initial) $(q_{\text{init}}, u) \cdots (q_n, u_n)$ est telle que $q_n \in F$ et $u_n = \epsilon$; c'est-à-dire dans une configuration acceptante. Autrement, si l'exécution est telle que $q_n \notin F$ et $u_n = \epsilon$, c'est-à-dire qu'elle termine dans une configuration non acceptante, on dit que le mot u n'est pas accepté.*

Définition 70 (Langage reconnu/accepté par un AFED) *Le langage reconnu par A , qu'on note $\mathcal{L}_{\text{auto}}(A)$, est l'ensemble $\{u \in \Sigma^* \mid u \text{ est accepté par } A\}$.*

Définition 71 (Langage à états) *Un langage $L \subseteq \Sigma^*$ est appelé langage à états, s'il existe un automate à nombre fini d'états et déterministe qui reconnaît L .*

Définition 72 (Classe des langages à états) *La classe des langages à états sur un alphabet Σ , noté $EF(\Sigma)$ ou EF lorsque l'alphabet importe peu ou se déduit du contexte, est l'ensemble des langages à états.*

Définition 73 (Fonction de transition étendue aux mots) *À partir de δ , on définit la fonction de transition étendue aux mots δ^* . Soit $q \in Q$, on a :*

- $\delta^*(q, \epsilon) = q$,
- pour $u = a_1 \cdot a_2 \cdots a_n$, $\delta^*(q, u) = \delta\left(\dots \delta\left(\delta(q, a_1), a_2\right) \dots, a_n\right)$.

Définition 74 (Fonction de transition étendue aux mots - définition inductive) *À partir de δ , on définit la fonction de transition étendue aux mots δ^* :*

- $\delta^*(q, \epsilon) = q$, pour tout état $q \in Q$,
- $\delta^*(q, u \cdot a) = \delta(\delta^*(q, u), a)$, pour tout état $q \in Q$, mot $u \in \Sigma^*$ et symbole $a \in \Sigma$.

Définition 75 (Acceptation d'un mot par un AFED avec fonction de transition étendue) *Un mot $u \in \Sigma^*$ est accepté par A si $\delta^*(q_{\text{init}}, u) \in F$. Un mot $u \in \Sigma^*$ n'est pas accepté par A si $\delta^*(q_{\text{init}}, u) \notin F$.*

3.2.2 Accessibilité et coaccessibilité

Définition 76 (Accessibilité d'un état dans un AFED) *Un état $q \in Q$ est accessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q_{\text{init}}, u) = q$.*

Définition 77 (Coaccessibilité d'un état dans un AFED) *Un état $q \in Q$ est coaccessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q, u) \in F$.*

3.3 Exercices

3.3.1 Automate et langage reconnu

Exercice 14 (♠) — Représentation tabulaire d'un automate

Considérons l'alphabet $\{a, b\}$ et les AFED dans la figure 3.2.

1. Donner la représentation tabulaire de ces AFED. Vous pourrez vous limiter à deux ou trois automates pour vérifier que vous avez compris le principe.

Exercice 15 (♠) — Langage reconnu par un automate

Considérons l'alphabet $\{a, b\}$ et les AFED dans la figure 3.2.

1. Décrire en langage naturel les langages reconnus ces AFED.
2. Comment les descriptions trouvées dans la question précédente seraient-elles modifiées si nous avions considéré l'alphabet $\{a, b, c\}$?

Exercice 16 (♠) — Automate complet

Considérons l'alphabet $\{a, b\}$ et les AFED dans la figure 3.2.

1. Indiquer les automates complets.

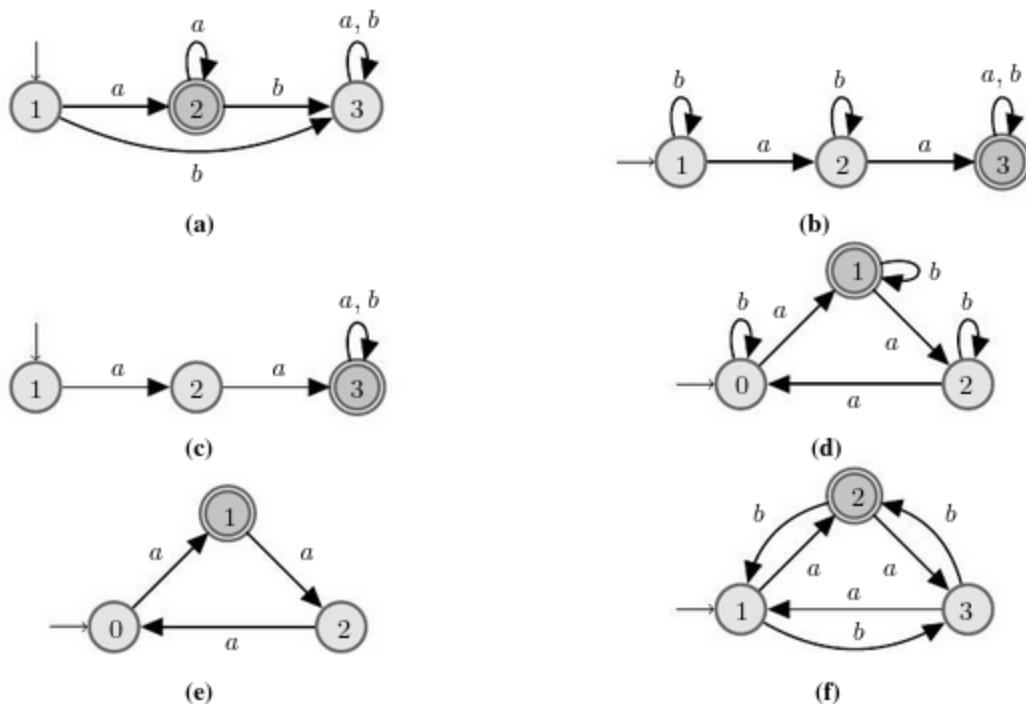


Figure 3.2 – Des automates déterministes.

Exercice 17 (♠♦) — Configuration et exécution

Considérons l'alphabet $\{a, b\}$ et les AFED dans la figure 3.2.

1. Pour chaque automate, donner un couple de configurations dans sa relation de dérivation (entre configurations).
2. Pour chaque automate, donner une exécution qui se termine dans une configuration terminale acceptante.
3. Pour chaque automate, donner une exécution qui se termine dans une configuration terminale non acceptante.
4. Pour chaque automate, donner une exécution qui se termine dans une configuration terminale de blocage, si cela est possible.

Exercice 18 (♠♦♣) — Dénombrement d'automates

Considérons l'alphabet $\Sigma = \{a, b\}$.

1. Combien il y a-t-il d'AFED complets différents avec 2 états ? Avec 3 états ?
2. Même question avec les AFED en général.

3.3.2 Automate reconnaissant un langage**Exercice 19 (♣)** — Langages avec des contraintes sur le nombre de symboles

Considérons l'alphabet $\{a, b\}$. Pour chacun des ensembles suivants, donner un AFED qui reconnaît cet ensemble de mots, si cela est possible.

1. L'ensemble des mots avec un nombre d'occurrences du symbole a multiple de 2.
2. L'ensemble des mots avec un nombre d'occurrences du symbole a multiple de 3.
3. L'ensemble des mots avec exactement n occurrences du symbole a , pour un certain $n \in \mathbb{N}$.
4. L'ensemble des mots avec n occurrences ou moins du symbole a , pour un certain $n \in \mathbb{N}$.
5. L'ensemble des mots avec strictement plus de n de occurrences du symbole a , pour un certain $n \in \mathbb{N}$.
6. L'ensemble des mots avec autant d'occurrences du symbole a que du symbole b .
7. L'ensemble des mots tels que chaque facteur de 3 symboles contienne exactement 2 occurrences du symbole a .

Exercice 20 (♣♦) — Langages avec certains préfixes, suffixes ou facteurs

Considérons l'alphabet $\{a, b, c\}$. Rappelons que \cdot désigne l'opérateur de concaténation entre mots (et langages). Pour chacun des langages suivants, donner un automate qui le reconnaît (si un tel automate existe).

1. L'ensemble des mots qui commencent par $a \cdot b$ ou $b \cdot c$.
2. L'ensemble des mots qui ne commencent ni par $a \cdot b$ ni par $b \cdot c$.

3. L'ensemble des mots qui contiennent $a \cdot b$.
4. L'ensemble des mots qui ne contiennent pas $a \cdot b$.
5. L'ensemble des mots qui terminent par $a \cdot b \cdot c$.
6. L'ensemble des mots qui ne terminent pas par $a \cdot b \cdot c$.
7. L'ensemble des mots de longueur supérieure ou égale à 2 et tels que l'avant-dernier symbole est b .

Exercice 21 (♠♠) — Langages avec des contraintes sur l'ordre des symboles

Considérons l'alphabet $\{a, b, c\}$. Pour les langages suivants, donner un automate reconnaisseur, si un tel automate existe.

1. L'ensemble des mots tels que a est toujours suivi de b .
2. L'ensemble des mots tels que a précède toujours b .
3. L'ensemble des mots tels que les occurrences du symbole a sont avant les occurrences du symbole b et les occurrences du symbole b sont avant les occurrences du symbole c .

Exercice 22 (♠♠) — Langage contenant des entiers

Considérons l'alphabet $\Sigma = \{1, 2, \dots, 9, 0\}$ et les entiers naturels notés en notation décimale usuelle.

1. Donner un AFED qui reconnaît les entiers naturels inférieurs à 245.

Exercice 23 (♠♠♠) — Langages contenant des multiples d'entiers

Considérons l'alphabet $\Sigma = \{1, 2, \dots, 9, 0\}$ et les entiers naturels notés en notation décimale usuelle.

1. Donner des AFED qui reconnaissent les entiers multiples de 2, 3, 5, 9, 10, 25, 50, 100, 250, 1000.
2. Donner des AFED qui reconnaissent les entiers multiples de 3 en bases 10, 2 et 4.
3. Donner des AFED qui reconnaissent les entiers multiples de 3 en base quelconque.
4. Reprendre les deux questions précédentes avec les multiples de 5.
5. Donner un AFED qui reconnaît les entiers multiples de x en base y avec $x, y \in \mathbb{N}$.

Exercice 24 (♠♠) — Langages des horaires et des dates

Considérons l'alphabet $\Sigma = \{1, 2, \dots, 9, 0\}$.

1. Donner un AFED qui reconnaît un horaire donné sous la forme $H_1H_2 h : M_1M_2m$, avec $H_1, H_2, M_1, M_2 \in \Sigma$.
2. Donner un AFED qui reconnaît une date dans l'année donnée sous la forme J_1J_2 / M_1M_2 avec $J_1, J_2, M_1, M_2 \in \Sigma$.

3.3.3 Fonction de transition d'un automate

Dans cette section, nous utilisons une définition **inductive** des mots (qui est équivalente à la définition de mots sous forme d'application) :

- $\epsilon \in \Sigma^*$ est un mot sur Σ
- Si $u \in \Sigma^*$ et si $a \in \Sigma$, alors $u \cdot a \in \Sigma^*$ est un mot sur Σ .

Exercice 25 (♠♠♠) — Fonction de transition étendue

À partir de la fonction de transition δ (opérant sur un état et un symbole), nous avons défini la fonction de transition étendue δ^* (son extension aux mots) comme la fermeture réflexive et transitive de δ (définition 73 (p. 48)).

1. Rappeler/proposer une définition inductive de cette fonction. Donner un argument justifiant le bon fondement de votre définition.
2. Démontrer que : $\forall x, y \in \Sigma^*. \forall q \in Q. \delta^*(q, x \cdot y) = \delta^*(\delta^*(q, x), y)$.

Exercice 26 (♠♠) — Configuration bloquante

1. Définir la notion de configuration bloquante.
2. Démontrer que si un automate est complet, alors il n'a pas de configuration bloquante.

Exercice 27 (♠♠) — Fonction de transition et état puits

Considérons un AFED $(Q, \Sigma, q_{\text{init}}, \delta, F)$ et $q \in Q$ un état particulier de cet automate tel que $\forall s \in \Sigma. \delta(q, s) = q$.

1. Démontrer par induction que $\forall u \in \Sigma^*. \delta^*(q, u) = q$.

Exercice 28 (♠♠♠) — Fonction de transition et symbole puits

Considérons un AFED $(Q, \Sigma, q_{\text{init}}, \delta, F)$ et $s \in \Sigma$ un symbole particulier tel que $\forall q \in Q. \delta(q, s) = q$.

1. Démontrer que : $\forall n \in \mathbb{N}. \delta^*(q, s^n) = q$ où s^n est le mot formé en concaténant n occurrences du symbole s (c'est-à-dire $\underbrace{s \cdot s \cdots s}_{n \text{ fois}}$).
2. Démontrer que soit $\{s\}^* \subseteq \mathcal{L}_{\text{auto}}(A)$ soit $\{s\}^* \cap \mathcal{L}_{\text{auto}}(A) = \emptyset$.

Exercice 29 (♠♠♠) — Fonction de transition et états jumeaux

Considérons un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, \{q_f\})$, et, supposons que pour chaque symbole $s \in \Sigma$ nous avons $\delta(q_{\text{init}}, s) = \delta(q_f, s)$.

1. Démontrer que, pour n'importe quel mot $u \neq \epsilon$, nous avons $\delta^*(q_{\text{init}}, u) = \delta^*(q_f, u)$.
2. Démontrer que, si un mot non vide u est reconnu par A , alors u^k (le mot formé par k concaténations du mot u) est aussi dans $\mathcal{L}_{\text{auto}}(A)$ pour chaque entier strictement positif k . Ceci peut se formaliser comme suit :

$$\forall u \in \Sigma^* \setminus \{\epsilon\}. (u \in \mathcal{L}_{\text{auto}}(A) \implies \forall k \in \mathbb{N}^*. u^k \in \mathcal{L}_{\text{auto}}(A)).$$



3.3.4 Langage à états ou non

Exercice 30 (♠♠♠) — Langage à états ou non

Pour chacun des langages suivants, indiquer si c'est un langage à états ou non. Justifier de manière informelle votre réponse.

1. \emptyset .
2. $\{a, a \cdot b\}$.
3. Σ^* .
4. $\{a^p \cdot b^q \mid p = q \wedge p = 5\}$.
5. $\{a^n \cdot b^n \mid n \in \mathbb{N}\}$.
6. $\{a^p \cdot b^q \mid (p, q) \in \mathbb{N}^2\}$.
7. $\{a^p \cdot b^q \mid p \geq q\}$.
8. $\{a^p \cdot b^q \mid p \geq q, q \geq 5\}$.
9. $\{a^p \cdot b^q \mid p \neq q\}$.
10. $\{a^p \cdot b^q \mid p = q\}$.
11. $\{a^p \cdot b^q \mid p < q\}$.
12. $\{a^p \cdot b^q \mid p \equiv q \pmod{7}\}$.
13. $\{a^n \cdot b \cdot a^n \mid n \in \mathbb{N}\}$.
14. $\{a^p \cdot b^q \mid p \geq q, q \leq 2017\}$.
15. $\{a^p \cdot b^q \mid p \geq q, q \geq 2017\}$.
16. $\{a^n \mid n \in \mathbb{N} \text{ est premier}\}$.
17. $\{a^{2^n} \mid n \in \mathbb{N}\}$.
18. $\{b \cdot a \cdot b \cdot a^2 \cdot b \cdot a^3 \cdots b \cdot a^n \mid n \in \mathbb{N}\}$.
19. $\{w \cdot w \mid w \in \Sigma^*\}$.
20. $\{w \cdot w \cdot w \mid w \in \Sigma^*\}$.
21. $\{u \in \Sigma^* \mid u \text{ est un palindrome}\}$.

3.3.5 Accessibilité et coaccessibilité

Exercice 31 (♣) — Donner les états accessibles et coaccessibles

Nous considérons les automates de la figure 3.2.

1. Indiquer les états accessibles et coaccessibles.

Exercice 32 (♣♣) — Automate et états accessibles et coaccessibles

1. Donner un exemple d'automate qui possède des états accessibles et coaccessibles, des états accessibles mais non coaccessibles, non accessibles mais coaccessibles, non accessibles et non coaccessibles.

Remarque 11 Les chapitres suivants reviennent sur les notions d'accessibilité et de coaccessibilité.

3.3.6 Automates et langages particuliers

Exercice 33 (♠♠♠) — Sous et sur-automate et langage reconnu

Soit A un automate. On appelle sous-automate de A tout automate obtenu en enlevant un ou plusieurs états de l'ensemble des états de A , à l'exception de l'état initial, ou en supprimant une ou plusieurs transitions de la fonction de transition de A et sans changer la nature des états

ni l'état initial. On appelle sur-automate de A tout automate obtenu en ajoutant un ou plusieurs états de l'ensemble des états de A , en ajoutant une ou plusieurs transitions à la fonction de transition de A (tout en préservant le déterminisme de l'automate) ; le sur-automate a le même état initial et son ensemble d'états accepteurs est un sur-ensemble de l'ensemble des états accepteurs de A .

1. Étant donné un automate $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$, définir (formellement) à quelle condition un automate $A' = (Q', \Sigma', q'_{\text{init}}, \delta', F')$ est un sous-automate de A .
2. Démontrer que le langage reconnu par tout sous-automate de A est inclus dans $\mathcal{L}_{\text{auto}}(A)$.
3. Démontrer que le langage reconnu par tout sur-automate de A contient $\mathcal{L}_{\text{auto}}(A)$.

3.4 Indications pour résoudre les exercices

Indications pour l'exercice 14 (p. 49)

1. S'inspirer de l'exemple dans la figure 3.1.

Indications pour l'exercice 16 (p. 49)

1. Selon la définition 63, un automate est complet si pour chaque état, la fonction de transition est définie pour chaque symbole.

Indications pour l'exercice 17 (p. 50)

1. Utiliser la définition de la relation de dérivation entre configurations, définition 67 (p. 47).
2. Partir d'une configuration initiale et la dériver jusqu'à que le mot soit vide dans la configuration et l'état soit accepteur.
3. Partir d'une configuration initiale et la dériver jusqu'à que le mot soit vide dans la configuration et l'état soit non accepteur.
4. Pour obtenir une configuration de blocage, considérer les automates non complets et les états où il n'y a pas de transitions sur certains symboles.

Indications pour l'exercice 18 (p. 50)

1. Déterminer le nombre de choix possibles pour chacun des éléments d'un quintuplet définissant l'AFED.

Indications pour l'exercice 25 (p. 52)

1. Suivre les cas de la définition inductive des mots. Donner une définition par induction sur le mot : un cas correspond au mot vide ϵ et l'autre à un mot non vide de la forme $u \cdot a$ où u est un mot quelconque et a un symbole.

2. Faire une preuve par induction en distinguant les cas comme dans la définition inductive de la question précédente.

Indications pour l'exercice 27 (p. 52)

1. Faire une preuve par induction en suivant la définition inductive de mot.

Indications pour l'exercice 28 (p. 52)

1. Faire une preuve par récurrence sur la variable quantifiée universellement sur les entiers naturels.
2. Distinguer deux cas en fonction du fait que l'état initial soit accepteur ou non.

Indications pour l'exercice 29 (p. 52)

1. Faire une preuve par induction sur les mots de $\Sigma^* \setminus \{\epsilon\}$.
2. Faire une preuve par récurrence sur $k \in \mathbb{N}$.

Indications pour l'exercice 33 (p. 53)

1. Utiliser la notion d'exécution pour montrer que si un mot est accepté par le sous-automate alors il est accepté par l'automate initial.

3.5 Solutions des exercices

Remarque 12 Dans la représentation graphique des automates, dans ce chapitre et les suivants, nous autoriserons d'étiqueter les transitions d'un automate par un ensemble, pour représenter l'ensemble de transitions étiquetées par les éléments de l'ensemble.

3.5.1 Automate et langage reconnu

Solution de l'exercice 14 (page 49)

1. Nous utilisons la représentation où les états sont en colonnes, les symboles en lignes. Le symbole \downarrow (resp. $*$) marque l'état initial (resp. un état accepteur).
 - Pour l'automate dans la figure 3.2a, une représentation tabulaire possible est donnée dans la figure 3.3a.
 - Pour l'automate dans la figure 3.2b, une représentation tabulaire possible est donnée dans la figure 3.3b.

	$\downarrow 1$	2^*	3
a	2	2	3
b	3	3	3

(a) Automate de la figure 3.2a.

	$\downarrow 1$	2	3^*
a	2	3	3
b	1	2	3

(b) Automate de la figure 3.2b.

Figure 3.3 – Automates de l'exercice 14 représentés sous forme tabulaire.**Solution de l'exercice 15 (page 49)**

- Considérons l'automate dans la figure 3.2a. Cet automate reconnaît les mots non vides contenant uniquement le symbole a . En effet, l'état initial est non accepteur, ainsi le mot vide n'est pas accepté. De plus, le seul état accepteur est l'état 2 qui est atteint depuis l'état initial en prenant la transition étiquetée par le symbole a . Il y a une transition étiquetée par le symbole a depuis l'état 2 vers lui-même. Ainsi, les mots commençant par le symbole a et ne contenant que des a sont acceptés. Ce sont les seuls mots acceptés, car on peut remarquer que depuis n'importe quel état les transitions étiquetées par b mènent à l'état 3 qui est non accepteur et dont toutes les transitions mènent à lui-même (état puits).
- Considérons l'automate dans la figure 3.2b. Cet automate reconnaît les mots qui contiennent au moins deux occurrences du symbole a . Le seul état accepteur est l'état 3. Sur les états 1 et 2, les transitions sur b amènent au même état et les transitions sur le symbole a font passer l'automate dans les états 2 et 3 respectivement. Par ailleurs, une fois l'état 3 atteint, les transitions sur les symboles a et b laissent l'automate dans le même état (accepteur). Ainsi, on voit qu'un mot amène l'automate dans l'état 3 dès que celui-ci contient deux occurrences du symbole a et l'automate reste toujours dans cet état.
- Considérons l'automate dans la figure 3.2c. Cet automate reconnaît les mots qui commencent par deux occurrences du symbole a , autrement dit, les mots avec $a \cdot a$ comme préfixe. Cet automate est similaire à celui de la figure 3.2b à la différence qu'il n'y a pas de transition sur le symbole b sur les états 1 et 2. Ce dernier point impose que les mots ne puissent pas avoir b parmi leur deux premiers symboles.
- Considérons l'automate dans la figure 3.2d. Cet automate reconnaît les mots qui contiennent un nombre d'occurrences du symbole a égal à 1 modulo 3 et sans contrainte sur le nombre d'occurrence du symbole b . L'automate change d'état uniquement sur le symbole a (et reste dans le même état sur le symbole b). Le symbole a fait passer l'automate des états 0, 1, 2 vers les états 1, 2, 0, respectivement. On voit donc que l'automate compte le nombre de a modulo 3. Par ailleurs, le numéro de l'état indique directement le reste de la division par 3 du nombre d'occurrences de a dans le mot lu. L'état 1 étant le seul état accepteur, ceci nous donne le langage annoncé.
- Considérons l'automate dans la figure 3.2e. Cet automate reconnaît les mots qui contiennent un nombre d'occurrences du symbole a égal à 1 modulo 3 et sans occurrence du symbole b . Cet automate est similaire à celui dans la figure 3.2d, à

l'exception qu'il n'y a pas de transition sur le symbole b . Ainsi, la présence d'au moins une occurrence du symbole b dans un mot lu en entrée rend l'exécution de ce mot non définie (et le mot rejeté par l'automate).

- Considérons l'automate dans la figure 3.2f. Cet automate reconnaît l'ensemble des mots tels que le nombre d'occurrences du symbole a moins le nombre d'occurrences du symbole b soit égal à 1 modulo 3. Cet automate est similaire à celui de la figure 3.2d à l'exception du fait que les transitions sur le symbole b sont inversées par rapport aux transitions sur le symbole a . Ainsi, si les transitions sur le symbole a ajoutent 1 modulo 3 à l'état, les transitions sur le symbole b soustraient 1 modulo 3 à l'état.
2. Si nous avions considéré l'alphabet $\{a, b, c\}$, aucun des automates ne serait complet et sur chaque état, il manquerait une transition sur le symbole c . Ceci implique que l'exécution de l'automate à partir d'un état ne peut se faire avec un symbole c . Ainsi, aux descriptions précédentes des langages, il faudrait donner le nouvel alphabet et indiquer que les mots des langages ne doivent pas comporter d'occurrence du symbole c .

Solution de l'exercice 16 (page 49)

1. Les automates complets sont ceux représentés dans les figures 3.2a, 3.2b, 3.2d, 3.2f. Les autres automates (figures 3.2c et 3.2e) ne sont pas complets. Par exemple, l'automate dans la figure 3.2c n'est pas complet, car il n'a pas de transition définie depuis les états 1 et 2 sur le symbole b .

Solution de l'exercice 17 (page 50)

1. Nous donnons des exemples de configurations reliées par la relation de dérivation des automates.
 - Pour l'automate dans la figure 3.2a, nous avons $(2, ba) \rightarrow (3, a)$.
 - Pour l'automate dans la figure 3.2b, nous avons $(1, aab) \rightarrow (2, ab)$.
 - Pour l'automate dans la figure 3.2c, nous avons $(1, ab) \rightarrow (2, b)$.
 - Pour l'automate dans la figure 3.2d, nous avons $(2, b) \rightarrow (2, \epsilon)$.
 - Pour l'automate dans la figure 3.2e, nous avons $(0, aaaa) \rightarrow (1, aaa)$.
 - Pour l'automate dans la figure 3.2f, nous avons $(3, ab) \rightarrow (1, b)$.
2. Nous donnons des exemples d'exécutions qui se terminent dans des configurations terminales acceptantes.
 - Pour l'automate dans la figure 3.2a, l'exécution du mot aa est $(1, aa)(2, a)(2, \epsilon)$.
 - Pour l'automate dans la figure 3.2b, l'exécution du mot aba est $(1, aba)(2, ba)(2, a)(3, \epsilon)$.
 - Pour l'automate dans la figure 3.2c, l'exécution du mot aab est $(1, aab)(2, ab)(3, b)(3, \epsilon)$.
 - Pour l'automate dans la figure 3.2d, l'exécution du mot ba est $(0, ba)(0, a)(1, \epsilon)$.
 - Pour l'automate dans la figure 3.2e, l'exécution du mot a est $(0, a)(1, \epsilon)$.
 - Pour l'automate dans la figure 3.2f, l'exécution du mot bb est $(1, bb)(3, b)(2, \epsilon)$.

Ces mots sont acceptés par les automates correspondants.

3. Nous donnons des exemples d'exécutions qui se terminent dans des configurations terminales non acceptantes.

- Pour l'automate dans la figure 3.2a, l'exécution du mot ab est $(1, ab)(2, b)(3, \epsilon)$.
- Pour l'automate dans la figure 3.2b, l'exécution du mot ba est $(1, ba)(1, a)(2, \epsilon)$.
- Pour l'automate dans la figure 3.2c, l'exécution du mot ϵ est $(1, \epsilon)$.
- Pour l'automate dans la figure 3.2d, l'exécution du mot aa est $(0, aa)(1, a)(2, \epsilon)$.
- Pour l'automate dans la figure 3.2e, l'exécution du mot aa est $(0, aa)(1, a)(2, \epsilon)$.
- Pour l'automate dans la figure 3.2f, l'exécution du mot ba est $(1, ba)(3, a)(1, \epsilon)$.

Ces mots ne sont pas acceptés par les automates correspondants.

4. Nous donnons des exemples d'exécutions qui se terminent dans des configurations de blocage, lorsque cela est possible.

- Pour l'automate dans la figure 3.2a, il n'y a pas de configuration de blocage (car l'automate est complet).
- Pour l'automate dans la figure 3.2b, il n'y a pas de configuration de blocage (car l'automate est complet).
- Pour l'automate dans la figure 3.2c, l'exécution du mot ab est $(1, ab)(2, b)$.
- Pour l'automate dans la figure 3.2d, il n'y a pas de configuration de blocage.
- Pour l'automate dans la figure 3.2e, l'exécution du mot b est $(0, b)$.
- Pour l'automate dans la figure 3.2f, il n'y a pas de configuration de blocage.

Remarquons que seuls les automates complets n'ont pas d'exécution qui termine dans une configuration de blocage.

Solution de l'exercice 18 (page 50)

1. Supposons que les automates à deux états aient pour ensemble d'états Q_2 , avec $|Q_2| = 2$. Il y a deux choix possibles pour l'état initial. L'ensemble des transitions est donné par une fonction de l'ensemble $Q_2 \times \Sigma$ vers l'ensemble Q_2 . Il y a donc $|Q_2|^{|Q_2 \times \Sigma|} = |Q_2|^{|Q_2| \times |\Sigma|}$ fonctions de transition possibles (car, pour deux ensembles finis A et B , $|A \times B| = |A| \times |B|$). L'ensemble des états accepteurs est un sous-ensemble de Q_2 . Il y a donc $2^{|Q_2|}$ ensembles d'états accepteurs possibles. Au final, cela fait $2 \times |Q_2|^{|Q_2| \times |\Sigma|} \times 2^{|Q_2|} = 128$ automates (syntaxiquement différents avec deux états). Nous pouvons suivre le même raisonnement que dans la question précédente en remplaçant Q_2 par $|Q_3| = 3$ pour trouver 17496 automates syntaxiquement différents avec trois états.

2. En généralisant le raisonnement précédent, pour un automate à n états, nous trouvons

$$\underbrace{n}_{\text{choix}} \quad \times \quad \underbrace{n^{n \times |\Sigma|}}_{\text{état initial}} \quad \times \quad \underbrace{2^n}_{\text{transitions}} \quad \underbrace{2^n}_{\text{états finaux}}$$

automates syntaxiquement distincts.

3.5.2 Automate reconnaissant un langage

Solution de l'exercice 19 (page 50)

Les automates de cet exercice sont donnés dans la figure 3.4.

1. Un AFED qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a multiple de 2 est représenté dans la figure 3.4a.
2. Un AFED qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a multiple de 3 est représenté dans la figure 3.4b.
3. Un AFED qui reconnaît l'ensemble des mots qui contiennent exactement n occurrences du symbole a est représenté dans la figure 3.4c.
4. Un AFED qui reconnaît l'ensemble des mots qui contiennent n occurrences ou moins du symbole a est représenté dans la figure 3.4d.
5. Un AFED qui reconnaît l'ensemble des mots qui contiennent strictement plus de n occurrences du symbole a est représenté dans la figure 3.4e. Ici, nous donnons l'automate où nous faisons l'interprétation dans laquelle il doit y avoir strictement plus de n occurrences de a dans le mot.
6. Il n'est pas possible de trouver d'automate pour ce langage.

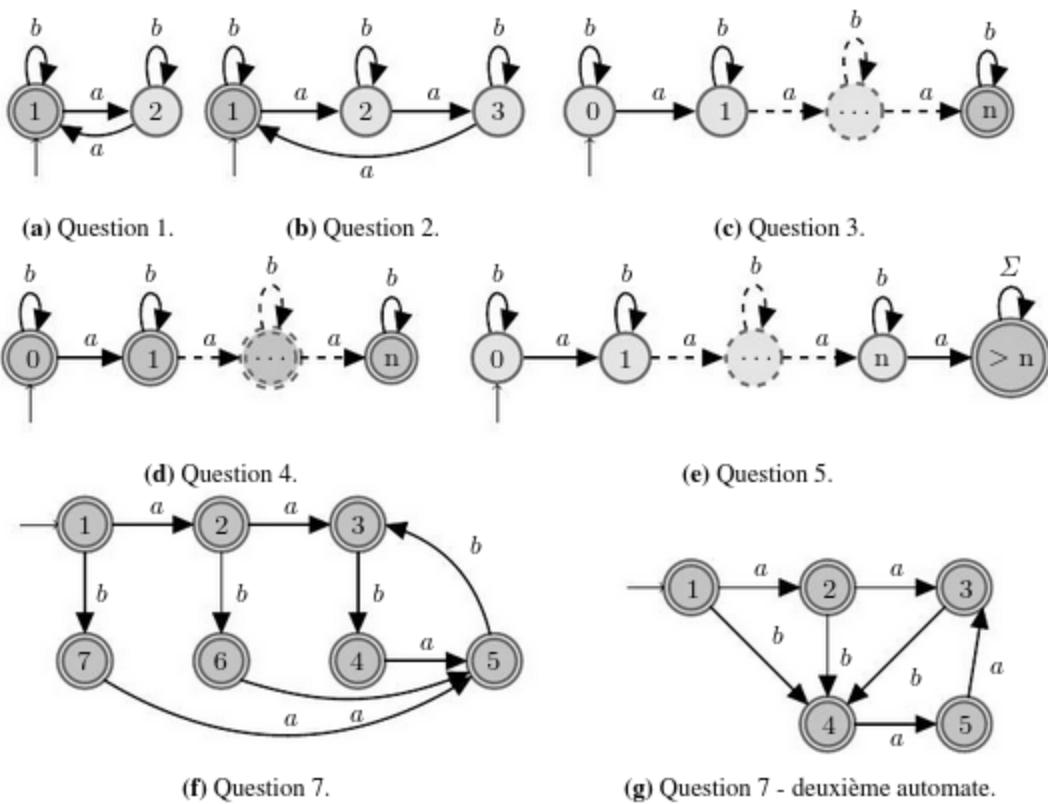


Figure 3.4 – Automates solutions de l'exercice 19 (p. 50).

7. Pour reconnaître l'ensemble des mots tels que chaque facteur de 3 symboles contienne (exactement) 2 occurrences du symbole a , nous proposons l'automate dans la figure 3.4f. Sur cet automate, les états 4, 6 et 7 peuvent être « fusionnés ». Nous obtenons ainsi l'automate dans la figure 3.4g.

Solution de l'exercice 20 (page 50)

Les automates de cet exercice sont donnés dans la figure 3.5.

- Pour reconnaître l'ensemble des mots qui commencent par $a \cdot b$ ou $b \cdot c$, nous proposons l'automate dans la figure 3.5a.

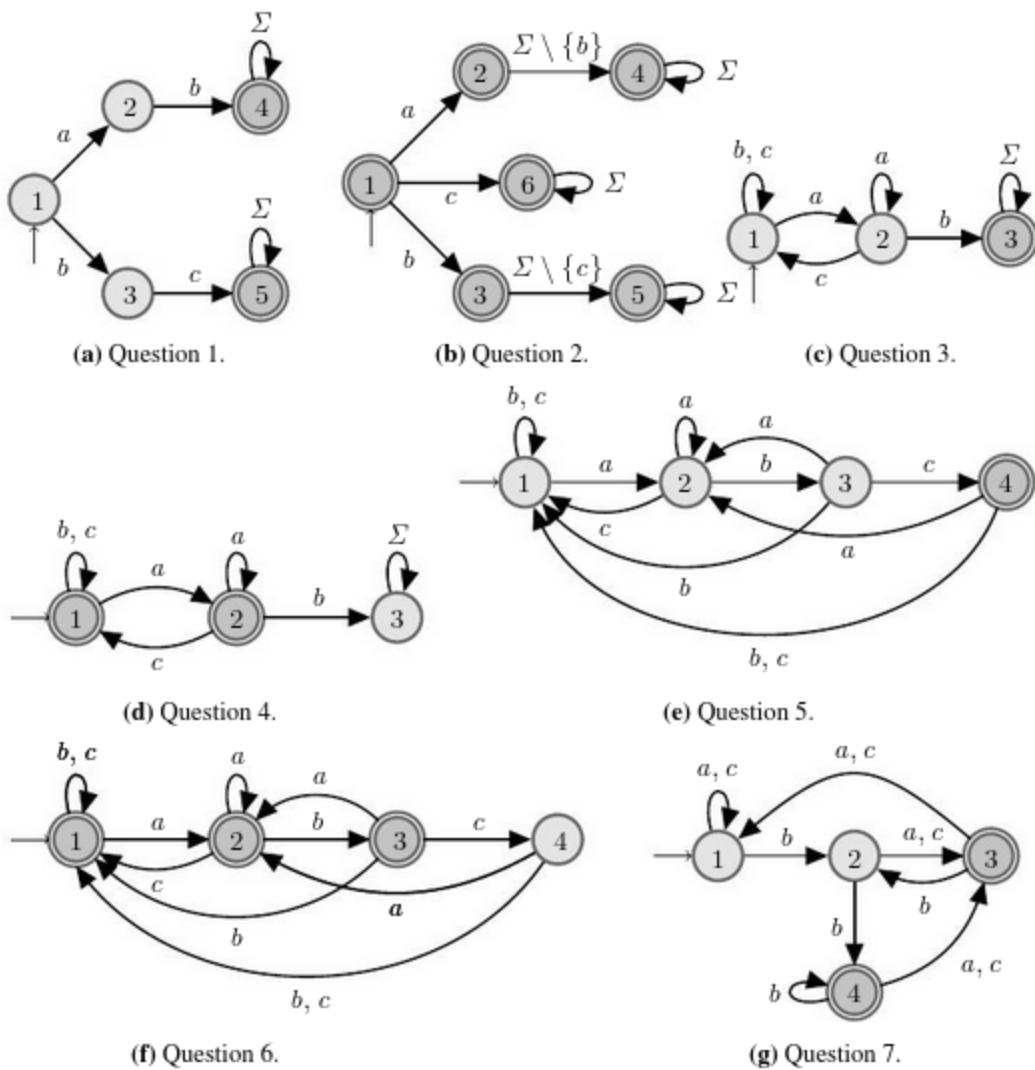


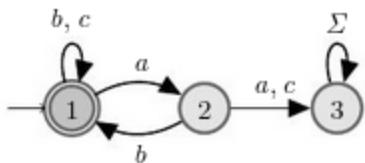
Figure 3.5 – Automates solutions de l'exercice 20 (p. 50).

2. Pour reconnaître l'ensemble des mots qui ne commencent ni par $a \cdot b$ ni par $b \cdot c$, nous proposons l'automate dans la figure 3.5b.
3. Pour reconnaître l'ensemble des mots qui contiennent (le facteur) $a \cdot b$, nous proposons l'automate dans la figure 3.5c.
4. Pour reconnaître l'ensemble des mots qui ne contiennent pas $a \cdot b$, nous proposons l'automate dans la figure 3.5d.
5. Pour reconnaître l'ensemble des mots qui terminent par $a \cdot b \cdot c$, nous proposons l'automate dans la figure 3.5e.
6. Pour reconnaître l'ensemble des mots qui ne terminent pas par $a \cdot b \cdot c$, nous proposons l'automate dans la figure 3.5f.
7. Pour reconnaître l'ensemble des mots de longueur supérieure ou égale à 2 et tels que l'avant-dernier symbole est b , nous proposons l'automate dans la figure 3.5g.

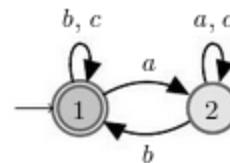
Solution de l'exercice 21 (page 51)

Soit $\Sigma = \{a, b\}$ l'alphabet des automates donnés ci-après. Les automates sont représentés dans la figure 3.6.

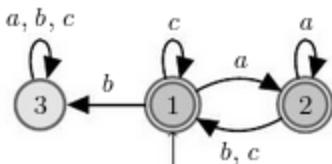
1. Pour reconnaître l'ensemble des mots tels que a est toujours suivi de b , il y a deux interprétations possibles. Dans la première, nous souhaitons que b soit le prochain



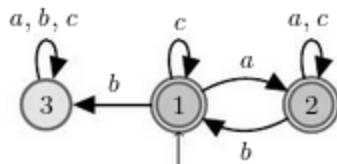
(a) Question 1. - première interprétation.



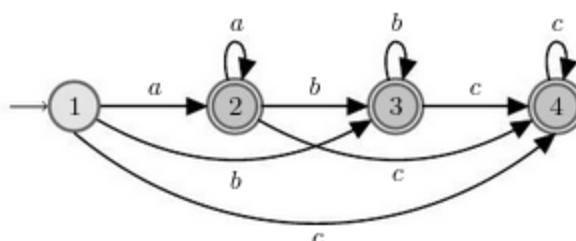
(b) Question 1. - seconde interprétation.



(c) Question 2. - première interprétation.



(d) Question 2. - seconde interprétation.



(e) Question 3.

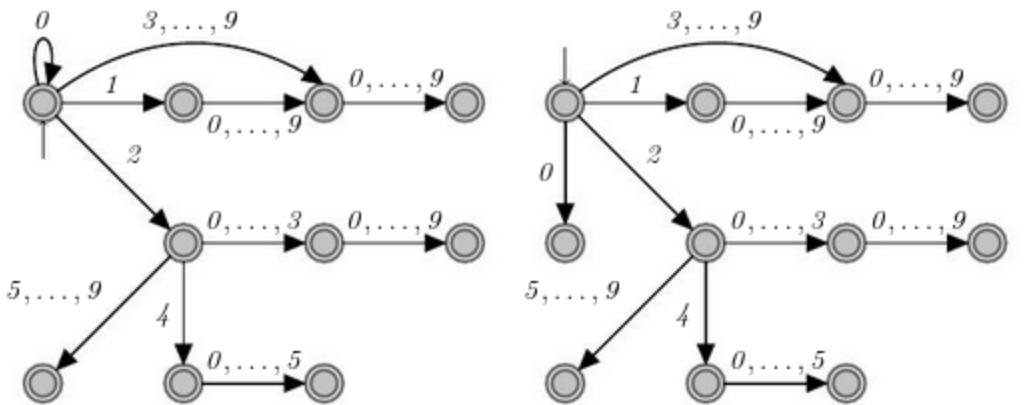
Figure 3.6 – Automates solutions de l'exercice 21 (p. 51).

symbole (automate dans la figure 3.6a). En particulier, les transitions sur a et c à partir de l'état 2 mènent à un état puits non accepteur. Dans la seconde, b n'est pas forcément le prochain symbole (automate dans la figure 3.6b) et les symboles a et c peuvent être lus depuis l'état 2 (qui reste non accepteur) jusqu'à la lecture du symbole b attendu.

2. Pour reconnaître l'ensemble des mots dans lesquels a précède toujours b , il y a deux interprétations possibles, de manière similaire à l'exercice précédent. Dans la première, le symbole a doit immédiatement précéder le symbole b (automate dans la figure 3.6c). Dans la seconde, a ne précède pas b forcément immédiatement (automate dans la figure 3.6d).
3. Un automate pour le langage est représenté dans la figure 3.6e.

Solution de l'exercice 22 (page 51)

1. Pour reconnaître les entiers naturels inférieurs à 245, nous proposons les automates dans la figure 3.7. L'automate dans la figure 3.7a accepte les entiers qui contiennent des chiffres non significatifs (et qui typiquement commencent par des 0). L'automate dans la figure 3.7b accepte uniquement les entiers qui contiennent des chiffres significatifs.



(a) Version qui accepte les entiers contenant des chiffres non significatifs.

(b) Version qui accepte les entiers contenant uniquement des chiffres significatifs.

Figure 3.7 – Automates qui reconnaissent les entiers naturels inférieurs à 245, solution de l'exercice 22 (p. 51).

Solution de l'exercice 23 (page 51)

1. Les automates suivants lisent des nombres, chiffre par chiffre (les symboles de l'alphabet sont des chiffres). L'idée commune à ces automates est d'obtenir un critère de reconnaissance sur les chiffres (ou une séquence de chiffres) à partir du critère sur le nombre. Les automates sont représentés dans la figure 3.8.
 - Pour reconnaître les nombres multiples de 2, nous proposons l'automate dans la figure 3.8a. Cet automate accepte les nombres dont le dernier chiffre lu est pair.
 - Pour reconnaître les nombres qui sont multiples de 3, nous proposons l'automate dans la figure 3.8b. Cet automate « compte la somme des chiffres lus modulo 3 ».

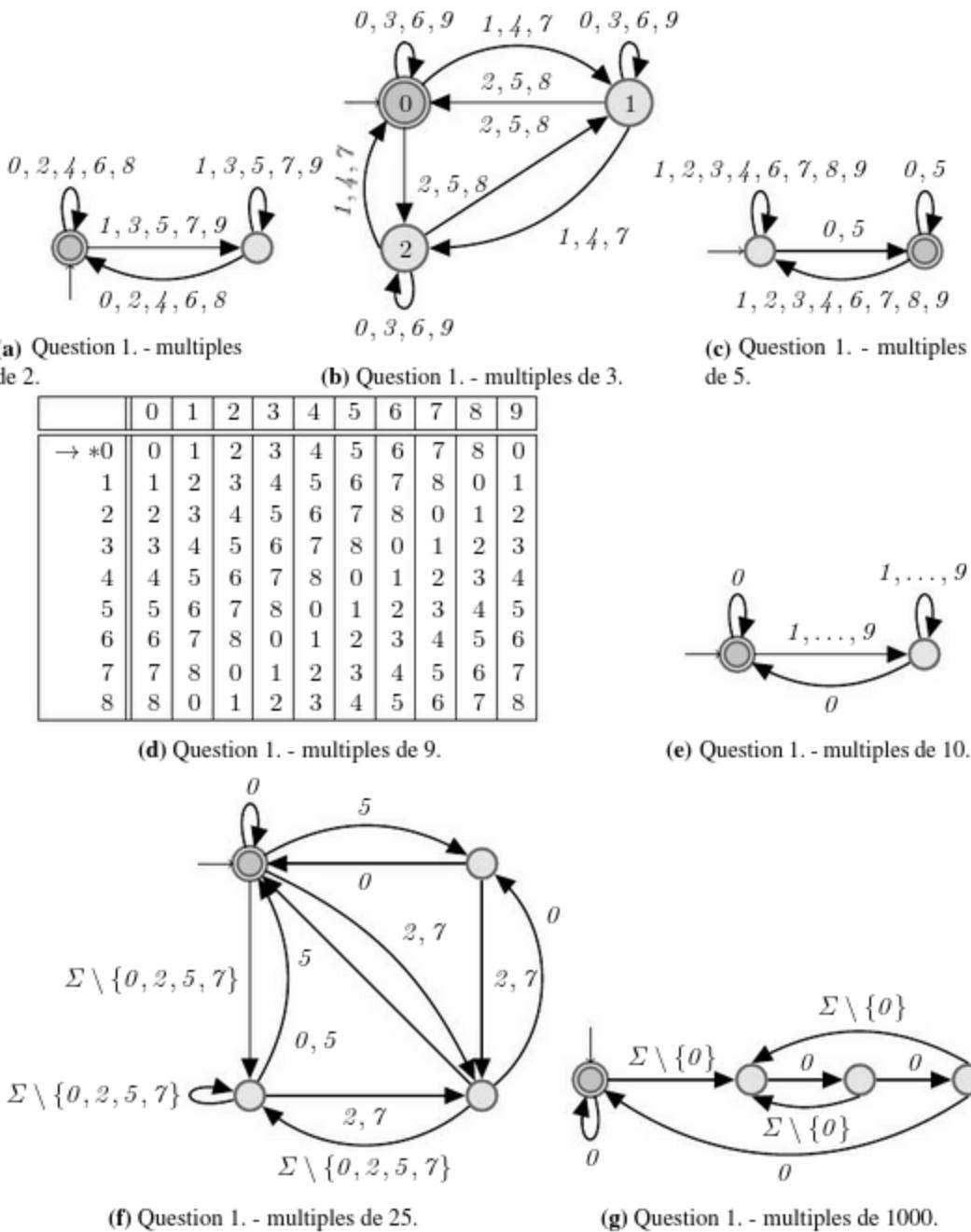


Figure 3.8 – Automates pour l'exercice 23 (p. 51)

En effet, un nombre en base 10 s'écrit de la manière suivante $\sum_i x_i \times 10^i$. Or, le reste 10 est congru à 1 modulo 3. Donc, le nombre est congru à $\sum_i x_i$ modulo 3. Il y a trois états dans l'automate, chacun correspondant à un reste de la division euclidienne du nombre lu par 3. L'état 0 est donc l'état initial. L'état 0 représente aussi la classe des nombres divisibles par 3 (congrus à 0 modulo 3). L'état 1 (resp. 2) représente la classe des nombres congrus à 1 (resp. 2) modulo 3. Lors de la lecture d'un chiffre x , le nombre de la classe est multiplié par 10 auquel on ajoute x . Or 10 est congru à 1 modulo 3. Effectuer une transition, revient donc à ajouter à l'état (élément de {0,1,2}) x et d'en faire la division par 3. Ainsi, si nous traitons en détail l'état 2, représentant $3k+2$, par les transitions étiquetées par :

- 0, 3, 6 et 9, on reste dans l'état ;
- 1, 4 et 7, on va dans l'état 0 ;
- 2, 5 et 8, on va dans l'état 1.

Par exemple, nous observons que 25 modulo 3 est bien égal à 1 (et le nombre lu jusqu'à présent peut se décomposer en $3 \times k + 1$, pour un certain $k \in \mathbb{N}$).

- Pour reconnaître les nombres multiples de 5, nous proposons l'automate dans la figure 3.8c qui accepte les nombres dont le dernier chiffre lu est 0 ou 5.
 - Pour les nombres multiples de 9, nous utilisons le même principe que pour l'automate reconnaissant les nombres multiples de 3. L'automate a 9 états, chacun de ces états est utilisé pour représenter le reste de la division du nombre courant par 9. L'automate est défini par la table de transition représenté dans la figure 3.8d (les états sont en lignes).
 - Pour vérifier si un nombre est multiple de 10, nous vérifions si le dernier chiffre est 0. Nous proposons l'automate dans la figure 3.8e.
 - Pour vérifier si un nombre est multiple de 25, nous vérifions si les deux derniers chiffres sont 00, 25, 50, ou 75. Nous proposons l'automate dans la figure 3.8f.
 - Pour vérifier si un nombre est multiple de 250, nous vérifions si les trois derniers chiffres sont 000, 250, 500, ou 750. L'automate s'obtient en suivant le principe utilisé pour l'automate de la question précédente.
 - Pour vérifier si un nombre est multiple de 1000, nous vérifions si les trois derniers chiffres sont 000. Nous proposons l'automate dans la figure 3.8g.
2. Pour trouver un automate reconnaissant les multiples de 3, nous utilisons, pour chaque base, un automate avec 3 états.
- Pour les nombres décrits en base 10, nous proposons l'automate représenté sur la figure 3.9a, sur l'alphabet $\{0, \dots, 9\}$. Nous remarquons que nous retrouvons l'un des automates trouvés dans la première question.
 - Pour les nombres décrits en base 2, nous proposons l'automate représenté sur la figure 3.9b, sur l'alphabet $\{0, 1\}$. En effet, suivant le principe utilisé dans les questions précédentes, comme les symboles décrivent un nombre en base 2, passer d'un état à un autre correspond à multiplier par 2 si le symbole lu est 0 et à multiplier par 2 et ajouter 1 si le symbole lu est 1.

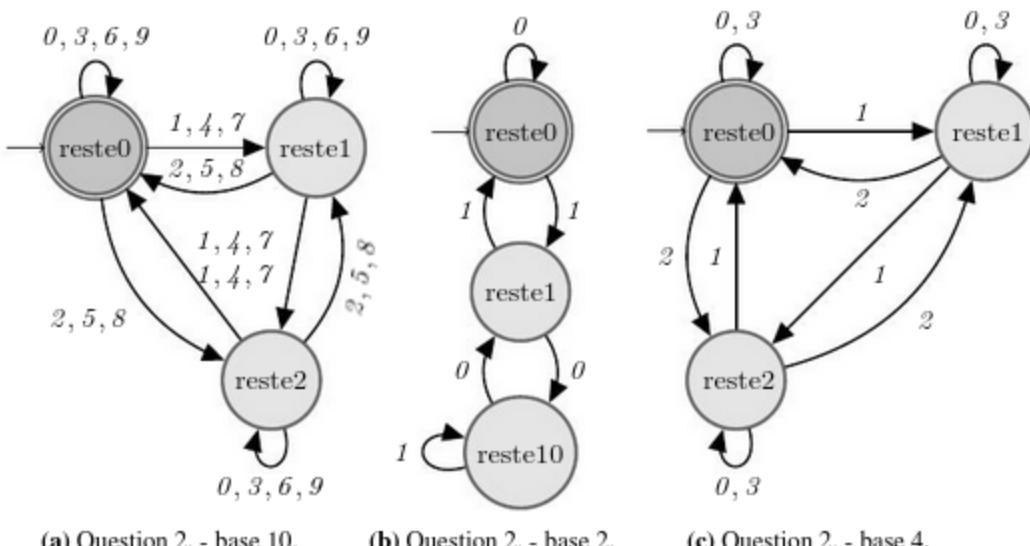
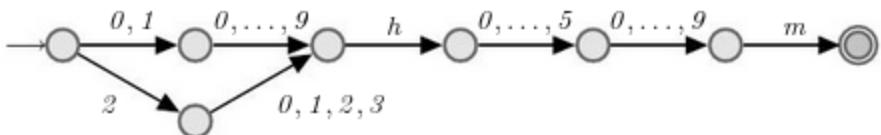


Figure 3.9 – Automates pour l'exercice 23 (p. 51)

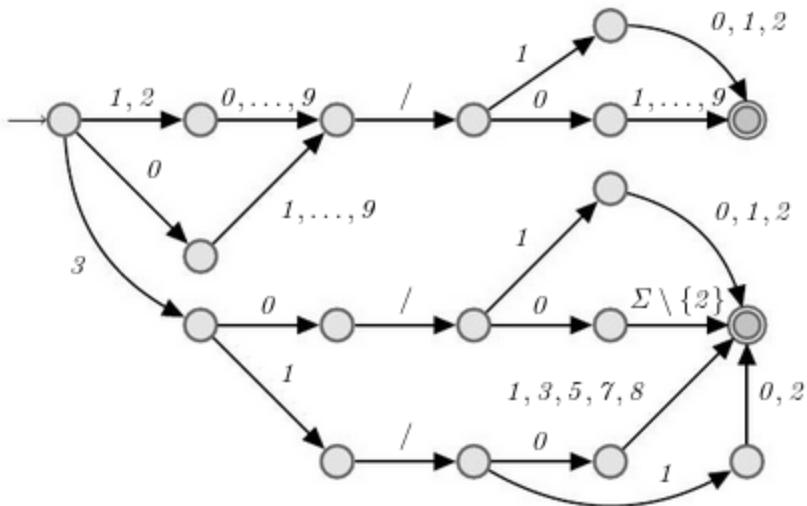
- Pour les nombres décrits en base 4, nous proposons l'automate dans la figure 3.9c, sur l'alphabet $\{0, 1, 2, 3\}$. Le principe est le même que pour les questions précédentes en modifiant la base, ce qui modifie l'effet de la lecture d'un symbole sur le nombre lu jusqu'à présent : la lecture d'un symbole $x \in \{0, 1, 2, 3\}$ à pour effet de multiplier le nombre lu jusqu'à présent par 4 et d'ajouter x .
3. Nous généralisons le raisonnement utilisé dans les questions précédentes. Nous utilisons toujours un automate à trois états (avec pour ensemble d'états $\{0, 1, 2\}$), correspondant aux restes de la division euclidienne par 3. L'alphabet est $\Sigma = \{0, \dots, x\}$ pour la base $x + 1$, avec $x \geq 0$. Depuis chaque état q , sur le symbole s , il y a une transition vers l'état $(q \times 3 + s) \bmod 3$.
 4. Le principe est le même qu'à la question précédente. Nous devons utiliser des automates à 5 états (qui comptent le reste de la division euclidienne par 5 du nombre lu).
 5. Nous généralisons en combinant les raisonnements utilisés jusqu'à présent. Pour l'automate reconnaissant les multiples de x en base y . L'ensemble des états est $Q = \{0, \dots, x - 1\}$. L'alphabet est $\Sigma = \{0, \dots, y - 1\}$. Depuis l'état $q \in Q$ sur le symbole $s \in \Sigma$, il y a une transition vers l'état $(q \times y + s) \bmod x$.

Solution de l'exercice 24 (page 51)

1. L'automate qui reconnaît les horaires dans le format donné est donné dans la figure 3.10a. L'automate d'abord reconnaît les heures puis ensuite les minutes.
2. L'automate qui reconnaît les dates possibles dans le format indiqué est donné dans la figure 3.10b. L'automate reconnaît les jours et ensuite les mois. Pour faciliter la représentation graphique, nous utilisons deux états accepteurs. Ces états accepteurs sont équivalents et peuvent être fusionnés.



(a) Automate pour la question 1. qui reconnaît les horaires.



(b) Automate pour la question 2. qui reconnaît les dates.

Figure 3.10 – Automates pour l'exercice 24 (p. 51).

3.5.3 Fonction de transition d'un automate

Solution de l'exercice 25 (page 52)

1. Nous définissons la fonction $\delta^* : Q \times \Sigma^* \rightarrow Q$ comme suit :

- $\delta^*(q, \epsilon) = q$, pour tout $q \in Q$, et
- $\delta^*(q, u \cdot a) = \delta(\delta^*(q, u), a)$, pour tout $u \in \Sigma^*$ et $a \in \Sigma$.

Cette définition est bien formée, car elle suit la définition inductive des mots sur Σ .

2. Soit $x \in \Sigma^*$, nous montrons la propriété par induction sur $y \in \Sigma^*$.

- **Cas de base.** Nous avons $y = \epsilon$. Dans ce cas, comme ϵ est l'élément neutre de la concaténation (voir exercice 5) $x \cdot y = x$. Alors, d'une part nous avons $\delta^*(q, x \cdot y) = \delta^*(q, x)$. D'autre part, comme $\delta^*(q, x) \in Q$, par définition de la fonction de transition étendue $\delta^*(\delta^*(q, x), \epsilon) = \delta^*(q, x)$.
- **Pas d'induction.** Soit $y \in \Sigma^*$, supposons que la propriété soit vérifiée pour le mot y . Soit $a \in \Sigma$, montrons que la propriété est vérifiée pour le mot $y \cdot a$. Soit $q \in Q$, par définition de la fonction de transition étendue, nous avons $\delta^*(q, x \cdot (y \cdot a)) = \delta(q, (x \cdot y) \cdot a) = \delta(\delta^*(q, x \cdot y), a)$. En utilisant l'hypothèse d'induction, nous avons $\delta(\delta^*(q, x \cdot y), a) = \delta(\delta^*(\delta^*(q, x), y), a)$. Comme $\delta^*(q, x) \in Q$, par définition de la fonction de transition étendue, nous avons $\delta(\delta^*(\delta^*(q, x), y), a) = \delta^*(\delta^*(q, x), y \cdot a)$.

Solution de l'exercice 26 (page 52)

1. Une configuration (q, u) est bloquante si $u \neq \epsilon$ et (q, u) n'a pas de successeur par la relation de dérivation.
2. Si l'automate est complet, alors il ne peut pas exister de configuration bloquante car $\forall q \in Q, \forall s \in \Sigma, \exists q' \in Q, \delta(q, s) = q'$.

Solution de l'exercice 27 (page 52)

1. Nous réalisons la preuve par induction en utilisant la définition inductive des mots.
 - **Cas de base.** Considérons le cas où $u = \epsilon$. Pour chaque $q \in Q$, nous avons : $\delta^*(q, \epsilon) = q$, par définition de la fonction de transition étendue.
 - **Pas d'induction.** Supposons que la propriété soit vérifiée pour un certain mot u et considérons le mot $u \cdot s$ pour un certain symbole $s \in \Sigma$. Calculons $\delta^*(q, u \cdot s)$. Par définition de la fonction de transition étendue, nous avons $\delta^*(q, u \cdot s) = \delta(\delta^*(q, u), s)$. En utilisant l'hypothèse d'induction, nous trouvons $\delta(\delta^*(q, u), s) = \delta(q, s) = q$.

Solution de l'exercice 28 (page 52)

1. Nous réalisons une preuve par récurrence sur n (qui est la variable quantifiée dans la proposition à démontrer). Nous avons deux cas.
 - **Cas de base.** Nous avons $n = 0$ et dans ce cas $s^n = \epsilon$. Par définition de la fonction de transition étendue, nous avons $\delta^*(q, \epsilon) = q$.
 - **Pas de récurrence.** Soit $n \in \mathbb{N}$, supposons que la propriété soit vraie. Considérons le mot $s^{n+1} = s^n \cdot s$. Nous avons $\delta^*(q, s^{n+1}) = \delta(\delta^*(q, s^n), s)$. En utilisant l'hypothèse de récurrence, nous avons $\delta^*(q, s^n) = q$ et donc $\delta^*(q, s^{n+1}) = \delta^*(q, s)$. Comme le mot s est de longueur 1, nous avons $\delta^*(q, s) = \delta(q, s)$. D'après la propriété de la fonction δ donnée dans l'énoncé, nous avons $\delta(q, s) = q$. Au final, nous trouvons $\delta^*(q, s^{n+1}) = q$.
2. $\mathcal{L}_{\text{auto}}(A)$ est le langage accepté par l'automate (qui est déterministe). Nous devons démontrer que soit le langage des mots formés d'une concaténation finie de symboles s est inclus dans $\mathcal{L}_{\text{auto}}(A)$, soit aucun mot formé d'une concaténation finie de s est dans $\mathcal{L}_{\text{auto}}(A)$. Ceci revient à démontrer que soit l'automate accepte tous les mots formés d'une concaténation finie de symboles s , soit il n'en accepte aucun.
Nous distinguons deux cas selon que l'état initial soit accepteur, c'est-à-dire selon que $q_{\text{init}} \in F$ ou pas.
 - Cas 1 : $q_{\text{init}} \in F$. D'après le résultat de la question précédente, nous avons : $\forall n \in \mathbb{N}, \delta^*(q_{\text{init}}, s^n) = q_{\text{init}}$. C'est-à-dire, tout mot formé par une concaténation de s est donc accepté par l'automate, et donc $\forall n \in \mathbb{N}, s^n \in \mathcal{L}_{\text{auto}}(A)$.
 - Cas 2 : $q_{\text{init}} \notin F$. En suivant un raisonnement similaire à celui de la question précédente, $\forall n \in \mathbb{N}, \delta^*(q_{\text{init}}, s^n) = q_{\text{init}}$ et donc $\forall n \in \mathbb{N}, s^n \notin \mathcal{L}_{\text{auto}}(A)$.

Solution de l'exercice 29 (page 52)

1. Soit $q \in Q$ l'état de A tel que $q = \delta(q_{\text{init}}, s) = \delta(q_f, s)$. Nous montrons la propriété par induction sur les mots de $\Sigma^* \setminus \{\epsilon\}$.

- **Cas de base.** La propriété est vérifiée par hypothèse pour chaque symbole $s \in \Sigma$.
 - **Pas d'induction.** Soit $u \in \Sigma^*$, $u \neq \epsilon$, un mot non vide. En utilisant le résultat de l'exercice 25, nous trouvons $\delta^*(q_{\text{init}}, s \cdot u) = \delta^*(\delta^*(q_{\text{init}}, s), u) = \delta^*(\delta(q_{\text{init}}, s), u) = \delta^*(q, u)$. De manière similaire, on trouve $\delta^*(q_f, s \cdot u) = \delta^*(q, u)$. Donc, on a bien $\delta^*(q_{\text{init}}, s \cdot u) = \delta^*(q_f, s \cdot u)$.
2. Soit $u \in \Sigma^*$ un mot tel que $u \in \mathcal{L}_{\text{auto}}(A)$. Prouvons par induction (récurrence) sur $k \in \mathbb{N}^*$ que $u^k \in \mathcal{L}_{\text{auto}}(A)$.
- **Cas de base.** Nous avons $k = 1$. Comme $u^1 = u$ et $u \in \mathcal{L}_{\text{auto}}(A)$, nous trouvons immédiatement que $u^1 \in \mathcal{L}_{\text{auto}}(A)$
 - **Pas d'induction.** Soit $k \in \mathbb{N}^*$, supposons que $u^k \in \mathcal{L}_{\text{auto}}(A)$. Considérons le mot u^{k+1} et montrons que $u^{k+1} \in \mathcal{L}_{\text{auto}}(A)$. Comme $u \in \mathcal{L}_{\text{auto}}(A)$ et q_f est l'unique état accepteur, nous avons $\delta^*(q_{\text{init}}, u) = q_f$. Comme $\delta^*(q_{\text{init}}, u) = \delta^*(q_f, u)$, nous avons $\delta^*(q_f, u) = q_f$. Examinons $\delta^*(q_{\text{init}}, u^{k+1})$. En utilisant le résultat de l'exercice 25, nous trouvons $\delta^*(q_{\text{init}}, u^{k+1}) = \delta^*(\delta^*(q_{\text{init}}, u^k), u) = \delta^*(q_f, u) = q_f$. Ceci signifie que u^{k+1} est accepté par l'automate, $u^{k+1} \in \mathcal{L}_{\text{auto}}(A)$.

Solution de l'exercice 30 (page 53)

Nous donnons des justifications intuitives sur le fait qu'un langage ne soit pas à états. Dans le chapitre 14, nous verrons une méthode permettant de le démontrer formellement.

1. Oui, ce langage est reconnu par tout AFED sans état accepteur.
2. Σ^* est le langage universel sur l'alphabet Σ . C'est un langage à état reconnu par un AFED dont l'état initial est accepteur et des transitions sur chaque symbole de Σ depuis l'état initial vers lui-même.
3. Oui, ce langage est fini. C'est donc un langage à états.
4. Oui, ce langage ne contient qu'un seul mot $a^5 \cdot b^5$ et est donc fini.
5. Non, ce langage n'est pas un langage à états. En effet, lors de lecture d'un mot de ce langage il faut « compter » le nombre (arbitraire) d'occurrences du symbole a qui ont été lus ; ceci afin de pouvoir déterminer si à la fin de la lecture si effectivement celui-ci comporte autant de a que de b .
6. Oui, ce langage peut s'écrire de manière équivalente $\{a^p \cdot b^q \mid p \in \mathbb{N} \wedge q \in \mathbb{N}\}$. C'est le langage des mots qui commencent par un nombre quelconque d'occurrences du symbole a suivies par un nombre quelconque d'occurrences du symbole b .
7. Non, ce langage n'est pas un langage à états. Nous pouvons utiliser également un argument de comptage. Un automate devrait pouvoir compter le nombre d'occurrences du symbole a qu'il a lu en entrée pour savoir combien d'occurrences du symbole b il peut lire en entrée. Autrement dit, pour chaque valeur de $p \in \mathbb{N}$, le comportement de l'automate sur la lecture des b est distinct.
8. Non, ce langage n'est pas un langage à états. L'argument est ici le même que pour la réponse à la question précédente, avec la contrainte supplémentaire que le nombre d'occurrences du symbole b doit être supérieure à 5.
9. Non, ce langage n'est pas un langage à états et nous pouvons utiliser de manière similaire un argument de comptage.

10. Non, ce langage n'est pas un langage à états et nous pouvons utiliser de manière similaire un argument de comptage.
11. Non, ce langage n'est pas un langage à états et nous pouvons utiliser de manière similaire un argument de comptage.
12. Oui, il suffit de faire un automate qui compte modulo 7 le nombre de a puis le nombre de b modulo 7 et compare le résultat. Écrire l'automate dans un chapitre suivant sera plus aisés avec l'utilisation des ϵ -transitions.
13. Non, ce langage n'est pas un langage à états et nous pouvons utiliser de manière similaire un argument de comptage.
14. Oui, ce langage est un langage à états. Pour chaque valeur de $p \leq 2017$, après avoir lu toutes les occurrences du symbole a , l'automate accepte un ensemble distinct de mots contenant que des occurrences du symbole b . Pour $p > 2017$, après avoir lu toutes les occurrences du symbole a , l'automate accepte l'ensemble des mots qui contiennent moins de 2017 occurrences du symbole b .
15. Non, ce langage n'est pas un langage à états. L'argument est ici le même que pour la réponse à la question 3, avec la contrainte supplémentaire que le nombre d'occurrences du symbole b doit être supérieure à 2017.
16. Non, ce langage n'est pas un langage à états. Intuitivement, pour ce langage, un automate aurait besoin de compter précisément le nombre de a qu'il a lu (avec des états distincts). Par ailleurs, le fait qu'il y ait une infinité de nombres premiers fait qu'il ne peut réaliser ce comptage avec un nombre fini d'états.
17. Non, ce langage n'est pas un langage à états. L'argument est le même que celui de la réponse précédente.
18. Non, ce langage n'est pas un langage à états. L'argument est le même que celui de la réponse précédente concernant le nombre d'occurrences du symbole a lues entre les occurrences du symbole b .
19. Non, ce langage n'est pas un langage à états. Pour chaque mot w de Σ^* , l'automate accepte un mot distinct uniquement : le mot w et a donc au moins un état distinct de l'ensemble des états utilisés après avoir lu un mot différent. Ainsi, un tel automate aurait besoin d'un nombre infini d'états distincts.
20. Non, ce langage n'est pas un langage à états. L'argument est le même que celui de la réponse précédente.
21. Non, ce langage n'est pas un langage à états. L'argument est le même que celui de la réponse précédente.

3.5.4 Accessibilité et coaccessibilité

Solution de l'exercice 31 (page 53)

1. Tous les états des automates dans la figure 3.2 sont accessibles. Tous les états des automates dans la figure 3.2 à l'exception de l'état 3 de l'automate dans la figure 3.2a sont coaccessibles.

Solution de l'exercice 32 (page 53)

1. Nous proposons l'automate dans la figure 3.11. Dans cet automate, l'accessibilité et la coaccessibilité des états est comme suit :

— accessibles : 0, 1, 2,	— coaccessibles : 0, 1, 3, 4
— non accessibles : 3, 4, 5,	— non coaccessibles : 2, 5

Ainsi, par exemple, l'état 1 est accessible et coaccessible, l'état 2 est accessible mais non coaccessible, l'état 3 est non accessible mais coaccessible et l'état 5 est non accessible et non coaccessible.

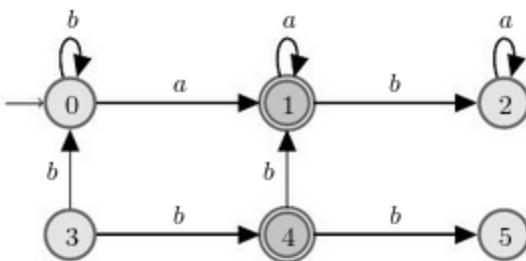


Figure 3.11 – Automate pour l'exercice 32.

3.5.5 Automates et langages particuliers

Solution de l'exercice 33 (page 53)

1. L'automate $A' = \left(Q', \Sigma', q'_{\text{init}}, \delta', F'\right)$ est un sous automate de l'automate $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ si : $Q' \subseteq Q$, $q'_{\text{init}} = q_{\text{init}}$, $\Sigma' \subseteq \Sigma$, $\delta' \subseteq \delta$ et $F' \subseteq F$.
2. Nous utilisons la notion d'exécution en montrant que si un mot est accepté par le sous-automate, alors il est accepté par l'automate initial. Considérons un mot accepté par le sous-automate. Son exécution est définie et acceptée par le sous-automate. Son exécution est une séquence de configurations qui nous indique certaines transitions entre certains états du sous-automate. Comme tous les états et les transitions appartiennent à l'automate initial, l'exécution de ce mot sur l'automate initial est la même séquence de configurations (avec le même état initial) qui est donc acceptée car la nature (accepteur ou non accepteur) des états n'est pas changée. Soient L et L' les langages reconnus par l'automate et le sous-automate, respectivement. Nous pouvons montrer $\forall u \in \Sigma'. \delta(q'_{\text{init}}, u) \in F' \implies \delta(q_{\text{init}}, u) \in F$ par induction sur u , en utilisant $\forall a \in \Sigma. \forall q_1, q_2 \in Q'. \delta'(q_1, a) = q_2 \implies \delta(q_1, a) = q_2$. Ceci implique que $\forall u \in \Sigma'. u \in L' \implies u \in L$.
3. Le résultat est une conséquence directe du résultat de la question précédente en remarquant que l'automate initial est un sous-automate du sur-automate.

Opérations sur les automates déterministes

4.1 Résumé intuitif du chapitre

Dans ce chapitre, nous nous intéressons aux opérations booléennes (complémentation, intersection et union) entre langages à états et aux opérations correspondantes sur les AFED. Aux opérations de compositions de langages (c'est-à-dire, d'ensembles de mots) sont associées des opérations sur leur automates reconnaiseurs.

Plus précisément, nous nous intéressons dans ce chapitre à répondre aux questions suivantes :

- Soit A un AFED et $\mathcal{L}_{\text{auto}}(A)$ son langage reconnu.
 - Le langage $\Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$ est-il reconnaissable par un AFED ?
 - Si oui, peut-on construire de manière effective un automate qui reconnaît $\Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$?
- Soient A et B deux AFED et $\mathcal{L}_{\text{auto}}(A), \mathcal{L}_{\text{auto}}(B)$, leur langages reconnus.
 - Le langage $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B)$ est-il reconnaissable par un AFED ?
 - Si oui, peut-on construire de manière effective un automate qui reconnaît $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B)$?
- Soient A et B deux AFED et $\mathcal{L}_{\text{auto}}(A), \mathcal{L}_{\text{auto}}(B)$ leur langages reconnus.
 - Le langage $\mathcal{L}_{\text{auto}}(A) \cup \mathcal{L}_{\text{auto}}(B)$ est-il reconnaissable par un AFED ?
 - Si oui, peut-on construire de manière effective un automate qui reconnaît $\mathcal{L}_{\text{auto}}(A) \cup \mathcal{L}_{\text{auto}}(B)$?

Dans le chapitre 8, nous étudierons d'autres opérateurs de composition de langages et d'automates, et étudierons les opérations de fermeture associées. Nous pourrons répondre de manière affirmative à toutes ces questions. Le fait que les réponses à ces questions soient positives

implique que les langages $\Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$, $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B)$ et $\mathcal{L}_{\text{auto}}(A) \cup \mathcal{L}_{\text{auto}}(B)$ sont des langages à états, car $\mathcal{L}_{\text{auto}}(A)$ et $\mathcal{L}_{\text{auto}}(B)$ sont des langages à états.

Ces propriétés sont appelées **fermeture de EF** par **complémentation, intersection et union**, respectivement.

Au-delà de l'aspect théorique, ces opérations ont une utilité pratique. En effet, elles permettent d'appréhender la construction d'automates, pour les langages à états, de manière *compositionnelle*. En effet, supposons que nous souhaitons trouver un automate reconnaisseur pour un langage à états qui s'exprime comme l'intersection de deux langages à états L_A et L_B , il est souvent plus simple de trouver un automate reconnaisseur pour L_A et un automate reconnaisseur pour L_B , puis d'appliquer l'opération associée à l'intersection de langages sur les automates.

Aussi, l'opération de complémentation, et la fermeture des langages à états par complémentation, seront aussi utiles pour démontrer que des langages ne sont pas à états. En effet, comme $\Sigma^* \setminus (\Sigma^* \setminus L) = L$, pour tout langage $L \subseteq \Sigma^*$, si $\Sigma^* \setminus L$ n'est pas un langage à états, alors nécessairement L ne l'est pas non plus. Pour certains langages, il sera plus facile de démontrer que leur complémentaire n'est pas à états. Nous étudierons comment montrer que des langages ne sont pas à états au chapitre 13.

Pour introduire l'opération de complémentation, nous serons amenés à introduire l'opération de **complétion** d'automates. Compléter un automate, par rapport à un alphabet donné en paramètre, ne change pas le langage reconnu par cet automate. En appliquant l'opération de complétion, la fonction de transition de l'automate devient définie sur chaque symbole de l'alphabet donné en paramètre, sans changer les transitions existantes.

4.2 Les notions essentielles

Nous considérons un alphabet Σ et un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ qui reconnaît un langage (noté $\mathcal{L}_{\text{auto}}(A)$).

Définition 78 (Complétion d'un AFED) *Le complété de A, noté C(A), est l'automate*

$$(Q \cup \{q_p\}, \Sigma, q_{\text{init}}, C(\delta), F)$$

tel que :

- $q_p \notin Q$ et
- $C(\delta) : Q \cup \{q_p\} \times \Sigma \rightarrow Q \cup \{q_p\}$ est une application définie par :

$$C(\delta)(q, a) = \begin{cases} \delta(q, a) & \text{si } (q, a) \in \text{domaine}(\delta), \\ q_p & \text{sinon.} \end{cases}$$

(Pour rappel q_p est appelé un « état puits », comme vu au chapitre 3.)

Proposition 1 (Correction de la procédure de complétion) *A et son complété reconnaissent le même langage :*

$$\mathcal{L}_{\text{auto}}(A) = \mathcal{L}_{\text{auto}}(C(A)).$$

Définition 79 (Complémentation d'un AFED complet) *Supposons l'automate A complet. Le complémentaire de A, noté A^c , est l'automate (complet) $(Q, \Sigma, q_{\text{init}}, \delta, Q \setminus F)$.*

Remarque 13 Pour que l'automate complémentaire reconnaise bien le langage complémentaire, il est indispensable que l'automate de départ soit complet.

Proposition 2 (Correction de la procédure de complémentation) Le langage reconnu par le complémentaire de A est le complémentaire du langage reconnu par A .

$$\mathcal{L}_{\text{auto}}(A^c) = \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A).$$

Corollaire 1 (Fermeture de EF par complémentation) La classe des langages à états EF est fermée par complémentation (ensembliste).

$$\forall L \subseteq \Sigma^*, L \in EF \implies \overline{L} \in EF.$$

Considérons deux AFED $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$.

Définition 80 (Produit de deux AFED) Le produit de A et B , noté $A \times B$, est l'automate $(Q, \Sigma, q_{\text{init}}, \delta, F)$ tel que :

- $Q = Q^A \times Q^B$,
- $q_{\text{init}} = (q_{\text{init}}^A, q_{\text{init}}^B)$,
- $\delta : (Q^A \times Q^B) \times \Sigma \rightarrow (Q^A \times Q^B)$ est telle que $\delta((q^A, q^B), s) = (\delta^A(q^A, s), \delta^B(q^B, s))$,
- $F = F^A \times F^B$.

Remarque 14 (Produit d'AFED avec des alphabets distincts) Si les alphabets des deux automates sont distincts, dans la définition de la fonction de transition, il faut distinguer deux cas.

- Dans le cas où le symbole appartient aux deux alphabets, la fonction de transition se comporte comme dans la définition précédente.
- Dans le cas où le symbole appartient à un seul des deux alphabets, il faut définir le comportement de la fonction de transition : le successeur peut être selon l'automate où le successeur est défini ou bien ne pas être défini.

Proposition 3 (Correction du produit de deux AFED) L'automate produit reconnaît l'intersection des langages reconnus par les automates à partir desquels il a été construit :

$$\mathcal{L}_{\text{auto}}(A \times B) = \mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B).$$

Corollaire 2 (Fermeture de EF par intersection) La classe des langages à états EF est fermée par intersection ensembliste.

$$\forall L_1, L_2 \subseteq \Sigma^*, (L_1 \in EF \wedge L_2 \in EF) \implies L_1 \cap L_2 \in EF.$$

Remarque 15 (Algorithmes associés aux opérations) Dans le chapitre 5 associé aux algorithmes, nous trouverons des algorithmes implémentant les définitions des opérations sur les AFED.

Corollaire 3 (Fermeture de EF par union) En conséquence du corollaire 1 et du corollaire 2, la classe des langages à états EF est fermée par union ensembliste ; car pour deux langages L_1 et L_2 , $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$.

4.3 Exercices

4.3.1 Automate complet / complété

Nous considérons la définition de l'automate complété (définition 78).

Exercice 34 (♠) — Complétude d'un automate

Considérons l'alphabet $\{a, b\}$ et les AFED dans la figure 3.2 (p. 49) du chapitre 3.

1. Déterminer quels automates sont complets.
2. Compléter les automates qui ne sont pas complets.
3. Compléter les automates en considérant l'alphabet $\{a, b, c\}$.

Exercice 35 (♠♠) — Compléter un automate complet

1. Que produit la complétion d'un automate complet ?
2. Modifier l'algorithme de complétion pour remédier au problème exhibé à la question précédente.

Exercice 36 (♠♠♠) — Correction de l'opération de complétion

Nous souhaitons démontrer que l'opération de complétion est correcte, c'est-à-dire qu'elle ne change pas le langage de l'automate sur lequel elle est appliquée. Étant donné un AFED A , $C(A)$ dénote l'automate résultant de l'application de l'opération de complétion à A .

1. Démontrer que $\mathcal{L}_{\text{auto}}(C(A)) = \mathcal{L}_{\text{auto}}(A)$.

4.3.2 Automate complémentaire

Exercice 37 (♠♠) — Trouver l'automate complémentaire

Considérons l'alphabet $\Sigma_1 = \{a, b, c\}$. Pour répondre aux questions suivantes, vous pouvez utiliser le résultat de l'exercice 36.

1. Pour chacun des automates de l'exercice 19 (p. 50) du chapitre 3, donner un automate qui reconnaît le complémentaire dans Σ_1^* du langage reconnu par l'automate.
2. Pour chacun des langages de l'exercice 20 (p. 50) du chapitre 3, donner un AFED qui reconnaît son complémentaire dans Σ_1^* (si cela est possible).

Exercice 38 (♠♠) — Trouver l'automate complémentaire

Considérons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui reconnaît le complémentaire des langages reconnus par les automates suivants.

1. L'automate représenté dans la figure 4.1a.
2. L'automate représenté dans la figure 4.1b.
3. L'automate représenté dans la figure 4.1c.

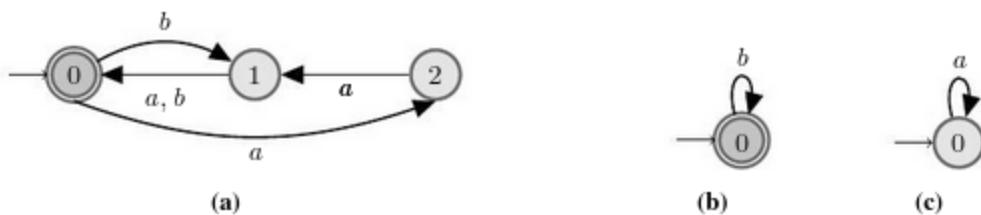


Figure 4.1 – Des automates déterministes.

Exercice 39 (♠♠♠) — Correction de l'opération de complémentation

Nous souhaitons démontrer que l'opération de complémentation est correcte, c'est-à-dire qu'elle produit bien un automate qui reconnaît le complémentaire de l'automate auquel elle a été appliquée (si l'automate de départ est complet). Nous considérons un AFED A complet sur un alphabet Σ . Nous notons A^c l'automate obtenu en appliquant l'opération de complémentation sur A .

1. Démontrer que $\mathcal{L}_{\text{auto}}(A^c) = \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$.

4.3.3 Automate produit

Exercice 40 (▲) — Obtenir un automate par produit d'automates

Considérons l'alphabet $\Sigma = \{a, b\}$ et les automates trouvés dans l'exercice 19 (p. 50).

1. Donner un AFED qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a multiple de 3 et multiple de 2.
 2. Donner un AFED qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a multiple de 2 et non multiple de 3.
 3. Pour chacun des langages précédents, donner un AFED qui reconnaît le langage complémentaire du langage reconnu dans Σ^* .

Exercice 41 (♠♠) — Obtenir un automate par produit d'automates

Considérons l'alphabet $\Sigma = \{a, b, c\}$. Pour chacun des langages suivants, en réutilisant les automates de l'exercice 20 (p. 50), donner un automate qui le reconnaît.

1. L'ensemble des mots qui ont $a \cdot b$ ou $b \cdot c$ pour préfixe et qui n'ont pas $a \cdot b \cdot c$ pour suffixe. (C'est-à-dire l'ensemble des mots qui commencent $a \cdot b$ ou $b \cdot c$ et qui ne terminent pas par $a \cdot b \cdot c$.)
 2. L'ensemble des mots qui contiennent un nombre pair d'occurrences du symbole c et qui ne contiennent pas le facteur $a \cdot b$.

Exercice 42 (♠♦) — L'opération de produit préserve la complétude

Nous considérons l'opération de produit entre automates (définition 80 (p. 73)).

1. Montrer cette opération préserve la complétude.

Exercice 43 (♠♠♠) — Démontrer qu'un langage est un langage à états

Soit L un langage à états sur un alphabet Σ .

1. Démontrer que l'ensemble des mots de L de longueur paire est un langage à états.
2. Démontrer que l'ensemble des mots de L ne contenant pas un certain symbole $s \in \Sigma$ est un langage à états.

Exercice 44 (♠♠♠) — Correction du produit d'automates

Soient $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ deux AFED. L'objectif de cet exercice est de démontrer que $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B) = \mathcal{L}_{\text{auto}}(A \times B)$.

1. Démontrer que pour chaque $n \in \mathbb{N}$, pour chaque exécution

- $(q_{\text{init}}^A, u_0) \cdot (q_1^A, u_1) \cdots (q_n^A, u_n)$ de A et
- $(q_{\text{init}}^B, u_0) \cdot (q_1^B, u_1) \cdots (q_n^B, u_n)$ de B

sur un mot commun u de longueur plus grande ou égale à n :

$$((q_{\text{init}}^A, q_{\text{init}}^B), u_0) \cdot ((q_1^A, q_1^B), u_1) \cdots ((q_n^A, q_n^B), u_n)$$

est une exécution de $A \times B$.

2. Utiliser le résultat précédent pour démontrer $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B) \subseteq \mathcal{L}_{\text{auto}}(A \times B)$.
3. Démontrer que $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B) \supseteq \mathcal{L}_{\text{auto}}(A \times B)$.
Nous pouvons en déduire que $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B) = \mathcal{L}_{\text{auto}}(A \times B)$.

4.4 Indications pour résoudre les exercices

Indications pour l'exercice 34 (p. 74)

1. Utiliser la définition 64 (p. 46) pour déterminer si un automate est complet (il doit y avoir une transition sur chaque symbole dans chaque état).
2. Introduire un état puits et ajouter une transition vers l'état puits depuis les états pour lesquels il manque une transition sur un symbole.

Indications pour l'exercice 35 (p. 74)

1. L'état puits q_p introduit lors de la complétion (définition 78 (p. 72)) n'est pas un état de l'automate de départ ($q_p \notin Q$). Que peut-on dire de cet état dans l'automate obtenu après l'opération de complétion ?

Indications pour l'exercice 36 (p. 74)

1. Montrer l'inclusion des langages l'un dans l'autre. Pour chacune des directions, raisonner sur l'exécution d'un mot accepté.

Indications pour l'exercice 37 (p. 74)

Utiliser la construction de l'automate complémentaire (définition 79), qui s'applique sur un AFED complet. Il est possible aussi d'utiliser le résultat de l'exercice 36.

Indications pour l'exercice 38 (p. 74)

Utiliser la construction de l'automate complémentaire (définition 79), qui s'applique sur un AFED complet. Il est possible aussi d'utiliser le résultat de l'exercice 36.

Indications pour l'exercice 39 (p. 75)

1. Montrer l'inclusion des langages l'un dans l'autre. Pour chacune des directions, raisonner sur l'exécution d'un mot accepté.

Indications pour l'exercice 42 (p. 75)

1. Considérer un état de l'automate produit et un symbole quelconques et montrer que la fonction de transition est définie pour cet état et symbole.

Indications pour l'exercice 43 (p. 76)

1. Trouver un automate pour le langage en utilisant la construction de l'automate produit.

Indications pour l'exercice 44 (p. 76)

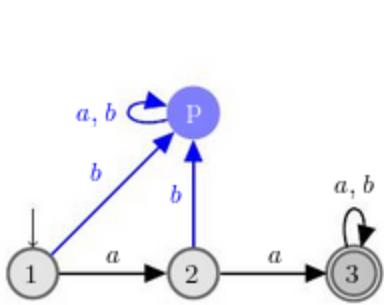
1. Faire une récurrence sur $n \in \mathbb{N}$. Dans le pas de récurrence, utiliser la définition de l'automate produit et la notion de dérivation entre configurations.
2. Utiliser le critère d'acceptation d'un mot sur les automates considérés.
3. À partir de l'exécution d'un mot sur le produit, déterminer l'exécution de ce mot sur chacun des deux automates impliqués dans le produit.

4.5 Solutions des exercices

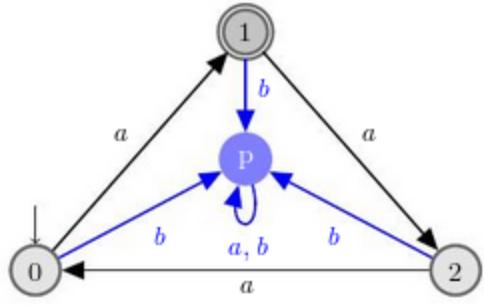
4.5.1 Automate complété

Solution de l'exercice 34 (page 74)

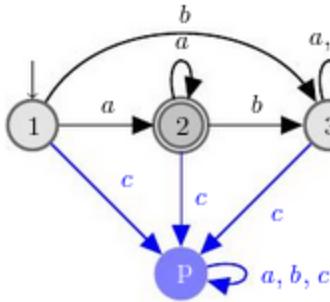
1. Les automates complets sont ceux représentés dans les figures 3.2a, 3.2b, 3.2d, 3.2f. Les autres automates (figures 3.2c et 3.2e) ne sont pas complets.
2. Nous complétons les automates indiqués comme non complets à la question précédente. Nous obtenons les automates représentés dans les figures 4.2a et 4.2b.
3. Nous complétons les automates en utilisant $\{a, b, c\}$ comme alphabet (de l'automate).



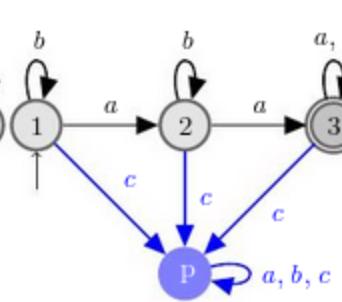
(a) Version complétée de l'AFED dans la figure 3.2c - alphabet $\{a, b\}$.



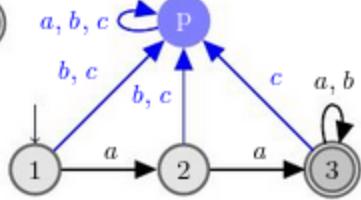
(b) Version complétée de l'AFED dans la figure 3.2e - alphabet $\{a, b\}$.



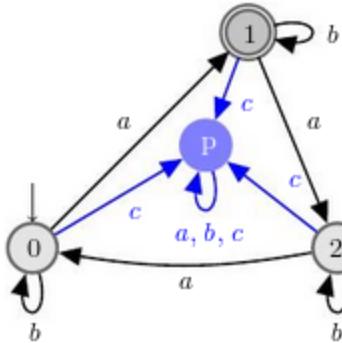
(c) Version complétée de l'AFED dans la figure 3.2a - alphabet $\{a, b, c\}$.



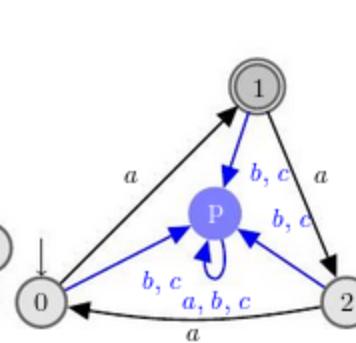
(d) Version complétée de l'AFED dans la figure 3.2b - alphabet $\{a, b, c\}$.



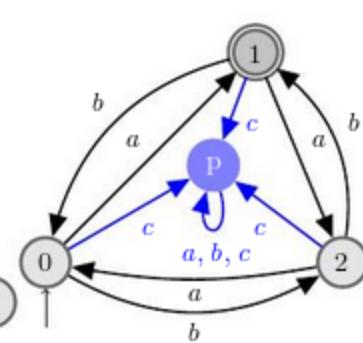
(e) Version complétée de l'AFED dans la figure 3.2c - alphabet $\{a, b, c\}$.



(f) Version complétée de l'AFED dans la figure 3.2d - alphabet $\{a, b, c\}$.



(g) Version complétée de l'AFED dans la figure 3.2e - alphabet $\{a, b, c\}$.



(h) Version complétée de l'AFED dans la figure 3.2f - alphabet $\{a, b, c\}$.

Figure 4.2 – Automates pour l'exercice 34 (p. 74).

Solution de l'exercice 35 (page 74)

- En appliquant la définition de l'automate complété (définition 78) sur un automate complet, nous introduisons un nouvel état qui est non accessible. En effet, l'algorithme introduit un nouvel état puits qui n'appartient pas à l'ensemble des états de l'automate. Comme l'automate de départ est complet, aucune transition ne sera ajoutée vers le nouvel état; celui-ci sera non accessible.
- Pour remédier au problème, il est possible de tester si l'automate est complet avant d'appliquer la transformation et d'appliquer celle-ci uniquement si l'automate n'est pas complet.

Solution de l'exercice 36 (page 74)

- Pour montrer $\mathcal{L}_{\text{auto}}(C(A)) = \mathcal{L}_{\text{auto}}(A)$, nous montrons $\mathcal{L}_{\text{auto}}(C(A)) \subseteq \mathcal{L}_{\text{auto}}(A)$ et $\mathcal{L}_{\text{auto}}(C(A)) \supseteq \mathcal{L}_{\text{auto}}(A)$. Pour la démonstration, nous considérons un mot $u = u_1 \cdots u_n$ de longueur n pour un $n \in \mathbb{N}$ sur l'alphabet Σ .
 - Démonstration de $\mathcal{L}_{\text{auto}}(C(A)) \subseteq \mathcal{L}_{\text{auto}}(A)$. Supposons que u soit accepté par $C(A)$, montrons que u est accepté par A . L'exécution de $C(A)$ sur u est telle que :

$$(q_1, u_1 \cdots u_n) \cdots (q_i, u_i \cdots u_n) \cdots (q_{n+1}, \epsilon),$$

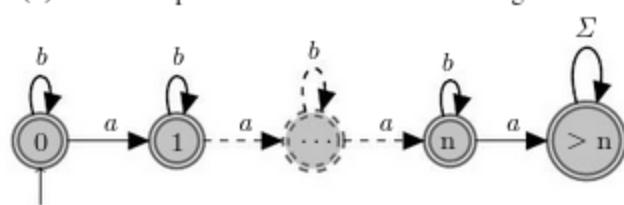
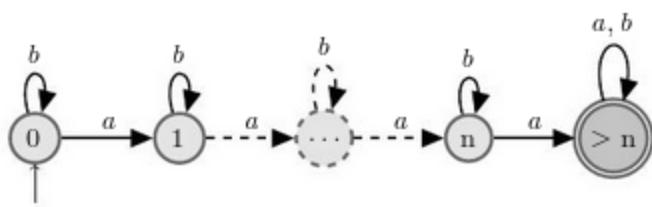
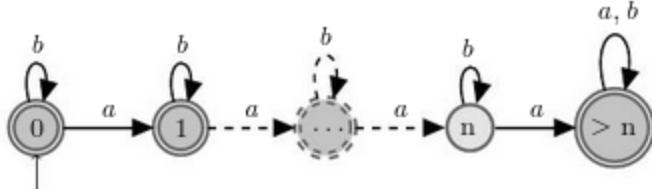
avec $\forall i \in [1, n] . q_i \in Q, q_1 = q_{\text{init}}$ et $q_{n+1} \in F$. Par l'absurde, il est possible de montrer que $\forall i \in [1, n+1] . q_i \neq q_p$, où q_p est l'état puits de $C(A)$. En effet, supposons qu'il existe i tel que $q_i = q_p$, alors pour $j \in [1, n+1], q_j = q_p$ et donc $q_{n+1} = q_p \notin F$. Ceci contredit le fait que $u \in \mathcal{L}_{\text{auto}}(C(A))$. L'exécution de $C(A)$ sur u est donc la même que celle de A sur u , et termine également dans un état terminal. Nous en déduisons que u est accepté par A .

- Démonstration de $\mathcal{L}_{\text{auto}}(C(A)) \supseteq \mathcal{L}_{\text{auto}}(A)$. Supposons que u soit accepté par A , montrons que u est accepté par $C(A)$. L'exécution de A sur u est telle que :

$$(q_1, u_1 \cdots u_n) \cdots (q_i, u_i \cdots u_n) \cdots (q_{n+1}, \epsilon),$$

avec $\forall i \in [1, n+1] . q_i \in Q, q_1 = q_{\text{init}}$ et $q_{n+1} \in F$. L'existence de l'exécution implique que $\forall i \in [1, n] . \delta(q_i, u_i) = q_{i+1}$, c'est-à-dire la fonction de transition est définie en q_i sur le symbole u_i , pour chaque $i \in [1, n]$. En utilisant la définition de la complétion, la fonction de transition de l'automate complété est telle que $C(\delta)(q_i, u_i) = q_{i+1}$. Par récurrence sur la longueur du mot, les états visités lors de l'exécution de A sur u sont les mêmes que les états visités lors de l'exécution de $C(A)$ sur u . L'exécution de $C(A)$ sur u est donc la même que celle de A sur u , et termine également dans un état terminal. Nous en déduisons que u est accepté par $C(A)$.

Nous pouvons en conclure que $\mathcal{L}_{\text{auto}}(C(A)) = \mathcal{L}_{\text{auto}}(A)$.



(f) AFED complémentaire de l'AFED dans la figure 3.4f.

Figure 4.3 – Automates pour la question 1. de l'exercice 37 (p. 74).

4.5.2 Automate complémentaire

Solution de l'exercice 37 (page 74)

Pour répondre aux questions suivantes, nous utilisons la construction de l'automate complémentaire (définition 79), qui s'applique sur un AFED complet. Cependant, nous utilisons le résultat de l'exercice 36 et nous introduisons un état puits dans un AFED uniquement si celui-ci n'est pas complet.

1. — L'AFED complémentaire qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a non multiple de 2 est représenté dans la figure 4.3a.
— L'AFED complémentaire qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a non multiple de 3 est représenté dans la figure 4.3b.
— Un AFED qui reconnaît l'ensemble des mots qui contiennent un nombre d'occurrences du symbole a différent de n est représenté dans la figure 4.3c.
— L'AFED complémentaire qui reconnaît l'ensemble des mots qui contiennent strictement plus de n occurrences du symbole a est représenté dans la figure 4.3d.
— Un AFED qui reconnaît l'ensemble des mots qui contiennent au plus n occurrences du symbole a est représenté dans la figure 4.3e.
— Un AFED qui reconnaît l'ensemble des mots tels que au moins un facteur de 3 symboles contienne 1 ou 3 occurrences du symbole a est représenté dans la figure 4.3f.
2. On remarque que les langages dans les questions 2., 4. et 6. sont les langages complémentaires des langages des questions 1., 3. et 5., respectivement. Cependant, ces automates ne sont pas ceux obtenus avec la construction de l'automate complémentaire, à partir de leur automate initial.
 - Pour reconnaître l'ensemble des mots qui n'ont pour préfixe ni $a \cdot b$ ni $b \cdot c$, l'AFED complémentaire obtenu est représenté dans la figure 4.4a.
 - Pour reconnaître l'ensemble des mots qui ont pour préfixe $a \cdot b$ ou $b \cdot c$, l'AFED complémentaire obtenu est représenté dans la figure 4.4b.
 - Pour reconnaître l'ensemble des mots qui ne contiennent pas (le facteur) $a \cdot b$, l'AFED complémentaire est représenté dans la figure 4.4c.
 - Pour reconnaître l'ensemble des mots qui contiennent (le facteur) $a \cdot b$, l'AFED complémentaire est représenté dans la figure 4.4d.
 - Pour reconnaître l'ensemble des mots qui ne terminent pas par (le suffixe) $a \cdot b \cdot c$, l'AFED complémentaire est représenté dans la figure 4.4e.
 - Pour reconnaître l'ensemble des mots qui terminent par (le suffixe) $a \cdot b \cdot c$, l'AFED est représenté dans la figure 4.4f.
 - Pour reconnaître l'ensemble des de longueur supérieure ou égale à 2 et tels que l'avant-dernier symbole n'est pas le symbole b , l'AFED complémentaire est représenté dans la figure 4.4g.

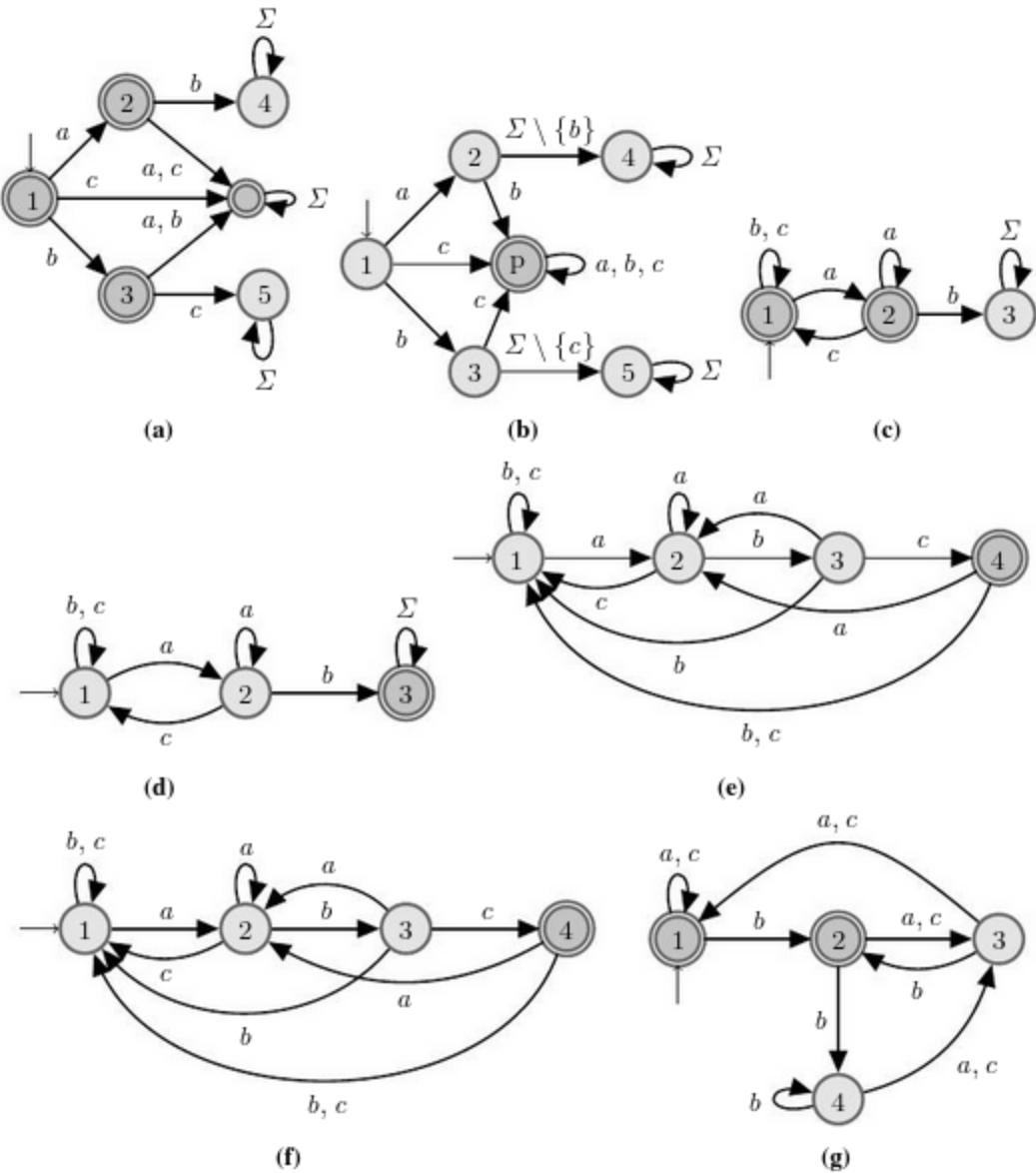
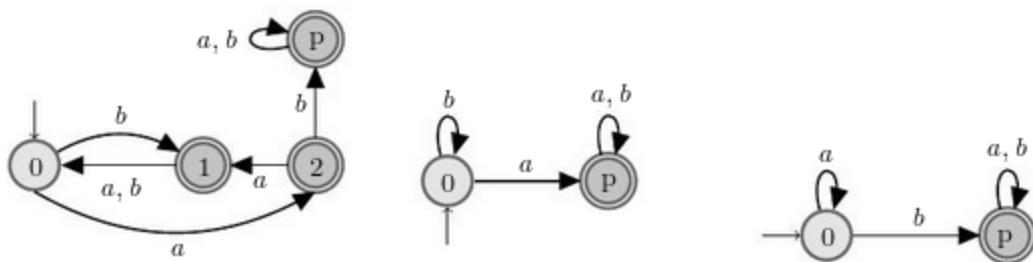


Figure 4.4 – Automates pour la question 2. de l'exercice 37 (p. 74).

Solution de l'exercice 38 (page 74)

1. L'AFED complémentaire de l'AFED dans la figure 4.1a est représenté dans la figure 4.5a.
2. L'AFED complémentaire de l'AFED dans la figure 4.1b est représenté dans la figure 4.5b.
3. L'AFED complémentaire de l'AFED dans la figure 4.1c est représenté dans la figure 4.5c.



(a) Automate pour la question 1. (b) Automate pour la question 2. (c) Automate pour la question 3.

Figure 4.5 – Automates pour l'exercice 38 (p. 74).

Solution de l'exercice 39 (page 75)

1. En utilisant les résultats de l'exercice 36, nous pouvons supposer que l'automate A est complet. L'exécution de A et A^c sur u s'écrit :

$$(q_1, u_1 \cdots u_n) \cdots (q_i, u_i \cdots u_n) \cdots (q_{n+1}, \epsilon),$$

avec :

- $\forall i \in [1, n] . q_i \in Q$,
- $q_1 = q_{\text{init}}$ et
- $\forall i \in [1, n] . \delta(q_i, u_i) = q_{i+1}$.

Pour montrer $\mathcal{L}_{\text{auto}}(A^c) = \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$, nous montrons que $\mathcal{L}_{\text{auto}}(A^c) \subseteq \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$ et $\mathcal{L}_{\text{auto}}(A^c) \supseteq \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$.

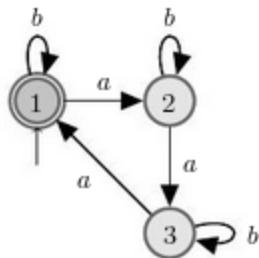
- Démonstration de $\mathcal{L}_{\text{auto}}(A^c) \subseteq \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$.

Supposons que u soit accepté par A^c , montrons que u n'est pas accepté par A . Comme u est accepté par A^c , nous avons $q_{n+1} \in Q \setminus F$, c'est-à-dire $q_{n+1} \notin F$. Nous en déduisons que u n'est pas accepté par A .

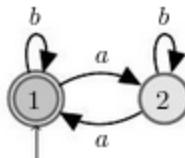
- Démonstration de $\mathcal{L}_{\text{auto}}(A^c) \supseteq \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$.

Supposons que u ne soit pas accepté par A , montrons que u est accepté par A^c . Comme u n'est pas accepté par A , nous avons $q_{n+1} \notin F$, c'est-à-dire $q_{n+1} \in Q \setminus F$. Nous en déduisons que u est accepté par A^c .

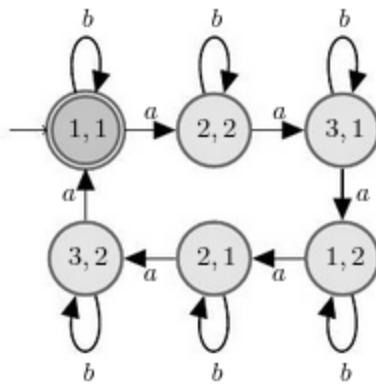
Nous en concluons que $\mathcal{L}_{\text{auto}}(A^c) = \Sigma^* \setminus \mathcal{L}_{\text{auto}}(A)$.



(a) Automate reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 3 – représentation graphique.



(b) Automate reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 2 – représentation graphique.



(c) Automate produit de l'automate dans la figure 4.6a et celui dans la figure 4.6b – représentation graphique.

	$\downarrow 1*$	2	3
a	2	3	1
b	1	2	3

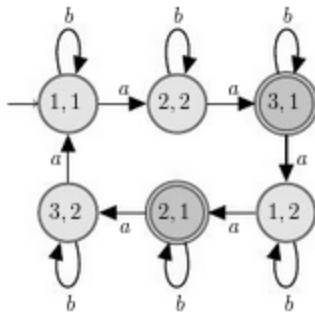
(d) Automate reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 3 – représentation tabulaire.

	$\downarrow 1*$	2
a	2	1
b	1	2

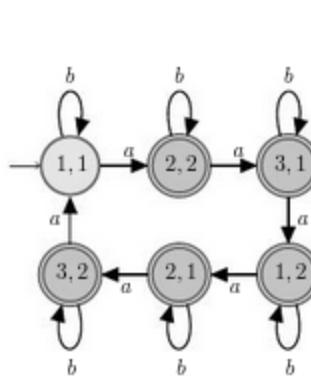
(e) Automate reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 2 – représentation tabulaire.

	$\downarrow 1, 1*$	1, 2	2, 1	2, 2	3, 1	3, 2
a	2, 2	2, 1	3, 2	3, 1	1, 2	1, 1
b	1, 1	1, 2	2, 1	2, 2	3, 1	3, 2

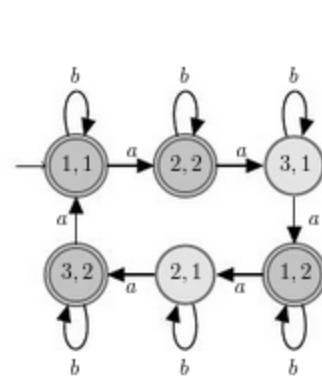
(f) Automate produit de l'automate dans la figure 4.6d et celui dans la figure 4.6e – représentation tabulaire.



(g) Automate reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 2 et non multiple de 3.



(h) Complémentaire de l'automate dans la figure 4.6c.



(i) Complémentaire de l'automate dans la figure 4.6g.

Figure 4.6 – Automates pour l'exercice 40 (p. 75). Pour les automates produits (de deux automates), nous utilisons x, y comme nom d'état pour l'état (x, y) .

4.5.3 Automate produit

Solution de l'exercice 40 (page 75)

- Nous construisons l'automate produit. Dans cette question, nous illustrons cette construction en utilisant les versions graphique et tabulaire des automates. Nous partons des automates reconnaissant les mots contenant un nombre d'occurrences du symbole a multiple de 3 et multiple de 2. Ces deux automates sont représentés dans les figures 4.6a et 4.6b, respectivement. Nous réalisons l'opération produit entre ces deux automates, pour obtenir l'automate représenté dans la figure 4.6c. En utilisant la version tabulaire des automates, nous représentons d'abord les automates sous forme tabulaire, puis nous faisons le produit. Ces automates sont représentés dans le tableau 4.6d, le tableau 4.6e et le tableau 4.6f, dans le même ordre que les automates représentés en version graphique.
- Nous réalisons l'opération produit entre les automates reconnaissant les mots avec un nombre d'occurrences du symbole a multiple de 2 et celui avec un nombre d'occurrences non multiple de 3 ; ce dernier étant le complémentaire de l'automate reconnaissant les mots avec un nombre d'occurrences de a multiple de 3. Nous obtenons l'automate représenté dans la figure 4.6g.
- Nous appliquons la construction de l'automate complémentaire sur les automates trouvés aux deux questions précédentes. Nous obtenons les automates représentés dans les figures 4.6h et 4.6i. Notons que les automates utilisés en entrée étant complets, nous n'appliquons pas l'opération de complétion.

Solution de l'exercice 41 (page 75)

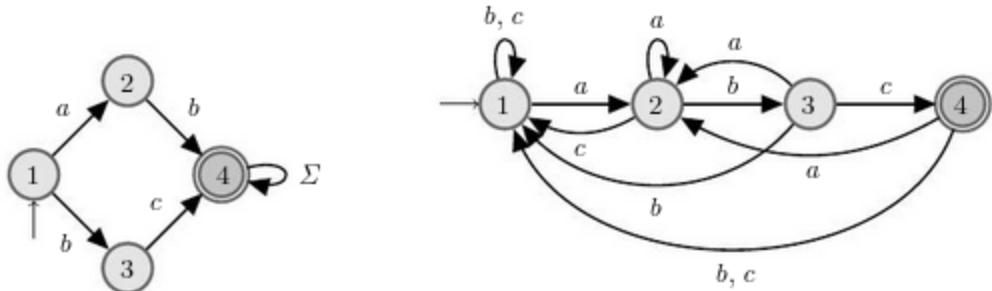
- Nous utilisons la version non complète des automates afin de réduire le nombre de transitions dans l'automate produit. Les automates utilisés sont représentés dans les figures 4.7a et 4.7b. Nous obtenons l'automate représenté dans la figure 4.7c.
- Nous utilisons la construction de l'automate produit sur l'automate qui reconnaît l'ensemble des mots qui contiennent un nombre pair d'occurrences du symbole c et sur l'automate qui reconnaît l'ensemble des mots qui ne contiennent pas le facteur $a \cdot b$. Ces automates ressemblent à ceux trouvés dans l'exercice 19. Nous obtenons l'automate représenté dans la figure 4.7d.

Solution de l'exercice 42 (page 75)

- Il s'agit de montrer qu'étant donnés deux AFED complets leur produit est un AFED complet.

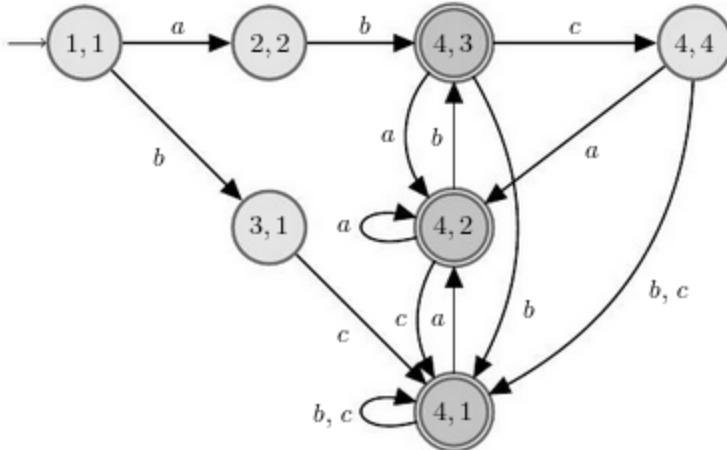
Soient $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ deux AFED complets, montrons que $A \times B = (Q, \Sigma, q_{\text{init}}, \delta, F)$ est complet, où $A \times B$ est l'automate produit de A et B obtenu selon la définition 80 (p. 73).

Soit $q \in Q$. Soit $s \in \Sigma$. Il s'agit de montrer que le successeur de q par s suivant δ existe. D'après la définition de l'ensemble d'états Q de $A \times B$, nous pouvons trouver deux états $q^A \in Q^A$ et $q^B \in Q^B$ tels que $q = (q^A, q^B)$. Comme A et B sont complets, les successeurs de q^A par s dans A et de q^B dans B sont définis, c'est-à-dire $\exists q^{A'} \in Q^A \cdot \delta^A(q^A, s) = q^{A'}$ et $\exists q^{B'} \in Q^B \cdot \delta^B(q^B, s) = q^{B'}$. Ainsi, le successeur de $q = (q^A, q^B)$ par s dans $A \times B$ existe et est $(\delta^A(q^A, s), \delta^B(q^B, s))$.

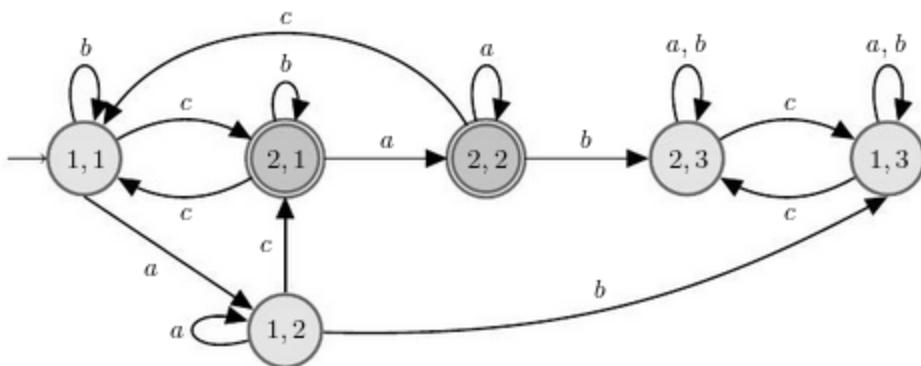


(a) Automate utilisé pour obtenir l'automate produit représenté dans la figure 4.7c.

(b) Automate utilisé pour obtenir l'automate produit représenté dans la figure 4.7c.



(c) Automate produit obtenu pour la question 1.



(d) Automate produit obtenu pour la question 2.

Figure 4.7 – Automates pour l'exercice 41 (p. 75). Pour les automates produits (de deux automates), nous utilisons x, y comme nom d'état pour l'état (x, y) .

d'après la définition de la fonction de transition de l'automate produit, c'est-à-dire $\delta((q^A, q^B), s) = (\delta^A(q^A, s), \delta^B(q^B, s))$, δ est définie dans l'état q sur le symbole s . Nous en concluons le résultat attendu.

Solution de l'exercice 43 (page 76)

1. Comme L est un langage à états, il existe un automate reconnaisseur A_L pour L . Par ailleurs, l'ensemble des mots de longueur paire sur l'alphabet Σ est un langage à états. Soit A_{pair} son automate reconnaisseur qui est représenté dans la figure 4.8a. Ainsi, pour l'ensemble des mots de L de longueur paire, nous pouvons trouver l'automate reconnaisseur obtenu par produit entre A_L et A_{pair} . Donc le langage est un langage à états, car on a exhibé un automate reconnaisseur.
2. De manière similaire, l'ensemble des mots sur Σ ne contenant pas un certain symbole est un langage à états et son automate reconnaisseur A_{sans} est représenté dans la figure 4.8b. Ainsi, pour l'ensemble des mots de L ne contenant pas le symbole $s \in \Sigma$, nous pouvons trouver l'automate reconnaisseur obtenu par produit entre A_L et A_{sans} ; c'est donc un langage à états.



(a) L'automate A_p reconnaissant les mots de longueur paire pour la question 1.

(b) L'automate A_{sans} reconnaissant les mots ne contenant pas un certain symbole pour la question 2.

Figure 4.8 – Automates pour l'exercice 43 (p. 76).

Solution de l'exercice 44 (page 76)

1. Montrons la propriété par récurrence sur $n \in \mathbb{N}$ (la longueur d'un mot).
 - Cas de base. Par définition de l'automate produit, l'état $(q_{\text{init}}^A, q_{\text{init}}^B)$ est l'état initial de l'automate produit. Pour ϵ , le mot de longueur 0, l'exécution est $(q_{\text{init}}^A, q_{\text{init}}^B)$. La propriété est donc vérifiée.
 - Pas de récurrence. Supposons la propriété vérifiée pour un certain $n \in \mathbb{N}$. Considérons les exécutions :

- $(q_{\text{init}}^A, u_0) \cdot (q_1^A, u_1) \cdots (q_n^A, u_n) \cdot (q_{n+1}^A, u_{n+1})$ de A sur u et
- $(q_{\text{init}}^B, u_0) \cdot (q_1^B, u_1) \cdots (q_n^B, u_n) \cdot (q_{n+1}^B, u_{n+1})$ de B sur u

sur un mot commun u de longueur plus grande ou égale à $n + 1$. Comme $(q_{\text{init}}^A, u_0) \cdot (q_1^A, u_1) \cdots (q_n^A, u_n) \cdot (q_{n+1}^A, u_{n+1})$ est une exécution de A sur u , $(q_0^A, u_0) \cdots (q_n^A, u_n)$ est également une exécution de A sur u . De manière similaire, $(q_{\text{init}}^B, u_0) \cdot (q_1^B, u_1) \cdots (q_n^B, u_n)$ est une exécution de B sur u . Comme u est un mot de longueur supérieure ou égale à $n + 1$, u est également un mot de

longueur supérieure ou égale à n . En utilisant l'hypothèse de récurrence, nous obtenons que $((q_{\text{init}}^A, q_{\text{init}}^B), u_0) \cdot ((q_1^A, q_1^B), u_1) \cdots ((q_n^A, q_n^B), u_n)$ est une exécution de $A \times B$ sur u . De plus, comme $(q_{\text{init}}^A, u_0) \cdot (q_1^A, u_1) \cdots (q_n^A, u_n) \cdot (q_{n+1}^A, u_{n+1})$ est une exécution de A , on obtient, en utilisant la définition de la notion d'exécution que $u_n = s \cdot u_{n+1}$ pour un certain $s \in \Sigma$ et que $\delta^A(q_n^A, s) = q_{n+1}^A$. De manière similaire, en utilisant le fait que $(q_{\text{init}}^B, u_0) \cdot (q_1^B, u_1) \cdots (q_n^B, u_n) \cdot (q_{n+1}^B, u_{n+1})$ est une exécution de B sur u , nous obtenons $\delta^B(q_n^B, s) = q_{n+1}^B$. Ainsi, en utilisant la définition de l'automate produit, nous obtenons $\delta((q_n^A, q_n^B), s) = (q_{n+1}^A, q_{n+1}^B)$. En utilisant de plus $u_n = s \cdot u_{n+1}$ et la notion de dérivation entre configurations pour l'automate produit, nous avons la dérivation $((q_n^A, q_n^B), u_n) \rightarrow ((q_{n+1}^A, q_{n+1}^B), u_{n+1})$ dans $A \times B$. Au final :

$$((q_{\text{init}}^A, q_{\text{init}}^B), u_0) \cdots ((q_n^A, q_n^B), u_n) \cdot ((q_{n+1}^A, q_{n+1}^B), u_{n+1}).$$

est une exécution de $A \times B$ sur u .

2. C'est une conséquence immédiate en utilisant le résultat de la question précédente et le critère d'acceptation d'un mot pour les automates A, B et $A \times B$. En effet, soit $u \in \mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B)$. Le mot u est accepté par A et par B . Nous en déduisons l'existence de deux exécutions : l'exécution $(q_{\text{init}}^A, u) \cdots (q_f^A, \epsilon)$ de A sur u et $(q_{\text{init}}^B, u) \cdots (q_f^B, \epsilon)$ de B sur u , avec $q_f^A \in F^A$ et $q_f^B \in F^B$. En utilisant le résultat de la question précédente, nous en déduisons l'exécution $((q_{\text{init}}^A, q_{\text{init}}^B), u) \cdots ((q_f^A, q_f^B), \epsilon)$ de $A \times B$ sur u . Comme $q_f^A \in F^A$ et $q_f^B \in F^B$, $(q_f^A, q_f^B) \in F^A \times F^B$ est un état accepteur de $A \times B$. Ainsi, u est accepté par $A \times B$. Donc, on a bien $\mathcal{L}_{\text{auto}}(A) \cap \mathcal{L}_{\text{auto}}(B) \subseteq \mathcal{L}_{\text{auto}}(A \times B)$.
3. Soit u un mot accepté par $A \times B$. Montrons que u est accepté par A . Comme u est accepté par $A \times B$, l'exécution de $A \times B$ sur u peut s'écrire $((q_{\text{init}}^A, q_{\text{init}}^B), u) \cdots ((q_f^A, q_f^B), \epsilon)$, avec $q_f^A \in F^A$ et $q_f^B \in F^B$. À partir de cette exécution, nous pouvons construire la séquence de configurations $(q_{\text{init}}^A, u) \cdots (q_f^A, \epsilon)$, obtenue en prenant l'exécution de $A \times B$ sur u et en projetant les états de $A \times B$ (composés d'un état de A et d'un état de B) sur l'état de A . Les mots dans la séquence de configurations sont les mêmes que ceux dans l'exécution de $A \times B$. Cette séquence de configurations est une exécution de A sur u , qui est acceptée. Montrer que u est accepté par B se réalise de manière similaire.

Algorithmes sur les automates déterministes

5.1 Résumé intuitif du chapitre

Dans ce chapitre, nous étudions les algorithmes sur les automates déterministes. Plus précisément, après avoir introduit les notions de **successeur** et **prédecesseur** d'un état, nous introduisons les notions de **parcours en largeur** et en **profondeur**.

Nous débutons par les algorithmes de **parcours** qui permettent de « visiter » les états de l'automate. Cette opération de visite permet de réaliser une certaine action sur un état ou d'y tester une condition. Les algorithmes de parcours permettent de réaliser une opération de visite sur tous les états de l'automate, sur un état particulier, un chemin particulier ou tous les chemins. Nous étudions deux stratégies de parcours, en largeur ou en profondeur. La stratégie de parcours est définie par une priorité dans le choix des successeurs. Dans le **parcours en largeur**, lorsqu'on se trouve dans un état q qui est le successeur d'un état q' , le parcours va privilégier la visite des autres successeurs de q' avant de visiter les successeurs de q . Dans le **parcours en profondeur**, c'est l'inverse. Nous définissons ces algorithmes de manière itérative, c'est-à-dire en itérant sur un ensemble de travail. Pour l'algorithme de **parcours en profondeur**, partant d'une **version récursive**, nous définissons une **version itérative**. La version récursive est plus intuitive mais peut poser des problèmes de performance à l'exécution sur des automates de grande taille.

Nous étudions ensuite les algorithmes pour le calcul des **états accessibles** et de calcul des **états coaccessibles**. Un état est accessible s'il peut être atteint à partir de l'état initial en suivant la fonction de transition. Un état est coaccessible si un état accepteur de l'automate peut être atteint à partir de cet état en suivant la fonction de transition.

Puis, nous étudions la **détection de cycles** dans un automate. Intuitivement, un automate possède un cycle s'il est possible de passer (au moins) deux fois par le même état en lisant un mot. Dit autrement, un cycle est une séquence de transitions consécutives dans la fonction de transition et telle que l'état de départ de la première transition soit l'état de d'arrivée de la

dernière transition. Nous noterons que le fait qu'un cycle autorise de passer deux fois par le même état implique qu'il permet également de passer autant de fois que l'on veut par cet état.

Enfin, nous étudierons quelques **problèmes de décision** liés aux automates. De manière générale, un problème de décision est une question, qui peut se formuler de manière précise, c'est-à-dire mathématiquement, et dont la réponse est oui ou non. Nous étudions deux problèmes de décision. Le premier est le **problème du langage vide** : est-ce que le langage reconnu par un automate est vide ? Le second est le **problème du langage infini** : est-ce que le langage reconnu par un automate est infini ?

Remarque 16 *Pour les algorithmes de ce chapitre, pour chaque algorithme, nous privilégions volontairement la version simple et intuitive ; sans forcément donner la version la plus efficace qui serait moins intuitive.*

Remarque 17 *Les algorithmes sont étudiés dans ce chapitre sur les automates déterministes, mais s'appliquent également aux automates non déterministes étudiés dans les prochains chapitres.*

5.2 Les notions essentielles

Dans la suite, nous utilisons un AFED $(Q, \Sigma, q_{\text{init}}, \delta, F)$ et un état $q \in Q$.

5.2.1 Successeurs et prédécesseurs d'un état

Nous introduisons les notions de successeur et prédécesseur d'un état qui servent à construire les autres notions de ce chapitre.

Définition 81 (Successeurs d'un état) *L'ensemble des états successeurs de $q \in Q$, selon δ , est :*

$$\text{Succ}(q) = \{q' \in Q \mid \exists a \in \Sigma. \delta(q, a) = q'\}.$$

Définition 82 (Prédécesseurs d'un état) *L'ensemble des états prédécesseurs de $q \in Q$, selon δ , est :*

$$\text{Pré}(q) = \{q' \in Q \mid \exists a \in \Sigma. \delta(q', a) = q\}.$$

Remarque 18 *Un état q est successeur d'un état q' si et seulement si q' est un prédécesseur de q .*

Dans la suite, nous supposons qu'il existe un ordre sur les symboles de l'alphabet. De plus, en itérant sur $\text{Succ}(q)$ ou $\text{Pré}(q)$, nous obtenons les états dans l'ordre suivant celui entre les symboles de l'alphabet.

5.2.2 Parcours

Les deux versions (réursive et itérative) suivantes de l'algorithme de parcours appliquent une opération de **visite**. Cette opération générique doit être précisée en fonction de l'objectif de l'algorithme. Par exemple, cette opération peut être d'afficher le nom de l'état, afficher les successeurs de l'état, enregistrer l'état dans un ensemble, etc. De plus, l'opération de visite des

états peut servir à construire le **résultat** de l'algorithme. Nous ne spécifions pas explicitement l'opération de visite ni le résultat. Le **résultat** peut être par exemple la liste des états selon l'ordre dans lesquels ils sont parcourus (avec une opération de visite qui ajoute l'état courant à la liste d'états) ou aucun résultat si l'on souhaite juste afficher le noms des états (avec une opération de visite qui réalise l'affichage).

Également, l'algorithme de parcours est paramétré par un ensemble d'états de départ qui est l'ensemble des états à partir desquels l'algorithme de parcours va commencer son exploration. Dans l'algorithme de calcul des états accessibles (algorithme 4 (p. 95)), cet ensemble est réduit au singleton contenant l'état initial de l'automate. Dans le cas où l'automate est un graphe non connexe, on pourra utiliser comme ensemble d'états de départ des états de parties non connectées de l'automate.

Définition 83 (Parcours - version récursive) *Étant donné un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$, un ensemble d'états Départ $\subseteq Q$ et une opération de « visite » s'appliquant à un état de l'automate, le parcours se fait avec l'algorithme 1 (p. 91) qui utilise l'algorithme 2 (p. 92). L'AFED A est global aux deux algorithmes. Le parcours récursif de l'automate se fait de manière récursive à partir de tous les états dans Départ pour y appliquer l'opération de visite. L'algorithme assure que l'opération de visite est appliquée au plus une fois sur chaque état.*

L'algorithme interne (algorithme 2) réalise l'opération de visite sur l'état q sur lequel il est appelé. L'algorithme s'appelle récursivement sur chacun des successeurs non visités de l'état présentement visité. Vérifier qu'un successeur n'a pas été déjà visité se fait juste avant l'appel récursif car un précédent appel récursif pour un autre successeur pourrait avoir réalisé la visite de cet état.

La terminaison de cet algorithme est assurée par le fait qu'un état est visité au plus une fois et que l'algorithme parcourir_rec n'est appelé sur un état que si cet état n'a pas été déjà visité (grâce à la variable Déjà_visités). De plus, chaque appel à parcourir_rec visite un état. Ainsi, on peut observer que le cardinal du sous-ensemble de Q (qui est fini) contenant les états non visités décroît strictement lors de chaque appel récursif.

Algorithme 1 parcourir() pour le parcours récursif à partir d'un ensemble d'états de départ

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$	(* un AFED *)
Entrée : Départ $\subseteq Q$	(* l'ensemble des états de départ du parcours *)
Sortie : résultat	(* construit par les visites d'états *)
ensemble d'états Déjà_visités;	(* variable globale aux deux algorithmes *)
Déjà_visités := \emptyset ;	(* initialement, rien n'est visité *)
pour chaque état $q \in$ Départ faire	
si $q \notin$ Déjà_visités alors	(* on vérifie que q n'a pas été déjà visité *)
parcourir_rec(q);	
fin si	
fin pour	
retourner résultat ;	(* si besoin, résultat dépend de la visite de chaque état *)

Remarque 19 *En pratique, l'ensemble Départ est réduit au singleton formé par l'état initial q_{init} de l'automate.*

Algorithme 2 *parcourir_rec()* pour le parcours à partir de l'état q

Entrée : $q \in Q$ (* un état *)

- 1: (* visiter l'état q *); (* dépend de l'objectif de l'algorithme *)
- 2: $\text{Déjà_visités} := \text{Déjà_visités} \cup \{q\}$; (* l'état q n'est plus à visiter *)
- 3: **pour** chaque état $q' \in \text{Succ}(q)$ **faire** (* pour chaque successeur q' de q *)
- 4: **si** $q' \notin \text{Déjà_visités}$ **alors** (* on vérifie que q' n'a pas été visité lors d'un parcours précédent *)
- 5: *parcourir_rec(q')*; (* on visite le successeur non déjà visité *)
- 6: **fin si**
- 7: **fin pour**

Définition 84 (Parcours - version itérative) Étant donnés un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$, un ensemble d'états Départ $\subseteq Q$ et une opération de « visite » s'appliquant à un état de A , le parcours se fait avec l'algorithme 3. Le parcours itératif de l'automate se fait de manière itérative à partir de tous les états dans Départ pour y appliquer l'opération de visite. L'algorithme assure que l'opération de visite est appliquée au plus une fois sur chaque état.

L'algorithme maintient des ensembles d'états Déjà_visités et À_visiter qui correspondent aux ensembles d'états déjà visités et aux états qui restent à visiter (car « découverts » par l'algorithme lors du parcours). Cet algorithme « consomme » les états enregistrés dans l'ensemble À_visiter et s'arrête lorsque celui-ci est vide. Lorsque l'algorithme sélectionne un état à visiter, il enregistre cet état comme n'étant plus à visiter, visite cet état (s'il n'a pas déjà été visité), l'enregistre comme déjà visité et ajoute l'ensemble des successeurs non déjà visités à l'ensemble des états à visiter.

La terminaison de cet algorithme est assurée car un état ne peut avoir été visité que s'il a été mis dans l'ensemble À_visiter. De plus, chaque état est visité au plus une fois : un état est mis dans À_visiter que s'il n'a pas déjà été visité et lors de la visite d'un état celui-ci est retiré de À_visiter. Chaque itération visite exactement un état. Ainsi, on peut observer que le cardinal du sous-ensemble de Q (qui est fini) contenant les états non visités décroît strictement lors de chaque appel récursif.

Remarque 20 (Stratégie de parcours) L'algorithme 3 utilise des ensembles (donc sans stratégie particulière) pour enregistrer les états à visiter. Si nous remplaçons la structure d'ensemble par une pile ou une file, nous définissons une stratégie de parcours.

- Si nous remplaçons l'ensemble d'états À_visiter par une file d'états¹, nous obtenons un parcours en largeur. Les instructions soit $q \in \text{À_visiter}$ et $\text{À_visiter} := \text{À_visiter} \setminus \{q\}$ sont remplacées par un défilage de la file À_visiter pour stocker l'élément obtenu dans q . L'instruction $\text{À_visiter} := \text{À_visiter} \cup (\text{Succ}(q) \setminus \text{Déjà_visités})$ est remplacée par un enfilage des éléments de $\text{Succ}(q) \setminus \text{Déjà_visités}$ dans la file À_visiter.

1. Une file est une structure de données où les éléments sont rangés en séquence et les éléments sont retirés de la file selon la politique premier inséré premier sorti.

Algorithme 3 *parcourir()* pour le parcours itératif de l'automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Entrée : Départ $\subseteq Q$ (* l'ensemble des états de départ du parcours *)

```

1: ensemble d'états  $\mathbb{A}_{\text{visiter}} := \text{Départ};$ 
2: ensemble d'états  $\text{Déjà\_visités} := \emptyset;$  (* initialement, rien n'est visité *)
3: tant que  $\mathbb{A}_{\text{visiter}} \neq \emptyset$  faire
4:   soit  $q \in \mathbb{A}_{\text{visiter}};$  (* prendre un état à visiter *)
5:    $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{q\};$  (* l'état  $q$  n'est plus à visiter *)
6:   si  $q \notin \text{Déjà\_visités}$  alors
7:     (* visiter l'état  $q$  *); (* dépend de l'objectif de l'algorithme *)
8:      $\text{Déjà\_visités} := \text{Déjà\_visités} \cup \{q\};$  (* l'état  $q$  vient d'être visité *)
9:      $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup (\text{Succ}(q) \setminus \text{Déjà\_visités});$ 
10:  fin si
11: fin tant que
12: retourner résultat; (* si besoin, résultat dépend de la visite de chaque état *)

```

- Si nous remplaçons l'*ensemble d'états* $\mathbb{A}_{\text{visiter}}$ par une *pile d'états*², nous obtenons un parcours en profondeur. Les instructions *soit* $q \in \mathbb{A}_{\text{visiter}}$ et $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{q\}$ sont remplacées par un dépilage de la pile $\mathbb{A}_{\text{visiter}}$ pour stocker l'élément obtenu dans q . L'instruction $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup (\text{Succ}(q) \setminus \text{Déjà_visités})$ est remplacée par un empilage des éléments de $\text{Succ}(q) \setminus \text{Déjà_visités}$ dans la pile $\mathbb{A}_{\text{visiter}}$.
- De manière similaire, si nous remplaçons l'*ensemble d'états* $\mathbb{A}_{\text{visiter}}$ par une *file d'états avec priorité*, nous obtenons l'algorithme de Dijkstra de calcul de plus court chemin.

5.2.3 Notions d'accessibilité et de coaccessibilité

Les notions d'accessibilité et de coaccessibilité et les calculs d'états accessibles et états coaccessibles s'obtiennent simplement à partir du parcours.

Accessibilité

Rappelons qu'un état $q \in Q$ est accessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q_{\text{init}}, u) = q$. Pour calculer les états accessibles avec un algorithme itératif, il faut passer à une définition inductive de la notion d'accessibilité.

Définition 85 (Ensemble des états accessibles) *L'ensemble des états accessibles dans A , noté Accessibles (A), est défini inductivement par :*

- l'*ensemble* $\{q_{\text{init}}\}$;
- la fonction $q \mapsto \text{Succ}(q)$.

2. Une pile est une structure de données où les éléments sont rangés en séquence et les éléments sont retirés de la file selon la politique dernier inséré premier sorti.

C'est-à-dire, le cas de base est que l'état initial q_{init} est accessible et le pas d'induction est que si $q \in Q$ est accessible dans A , alors tous les successeurs de q (états dans $\text{Succ}(q)$) sont accessibles dans A . La figure 5.1 illustre cette définition.

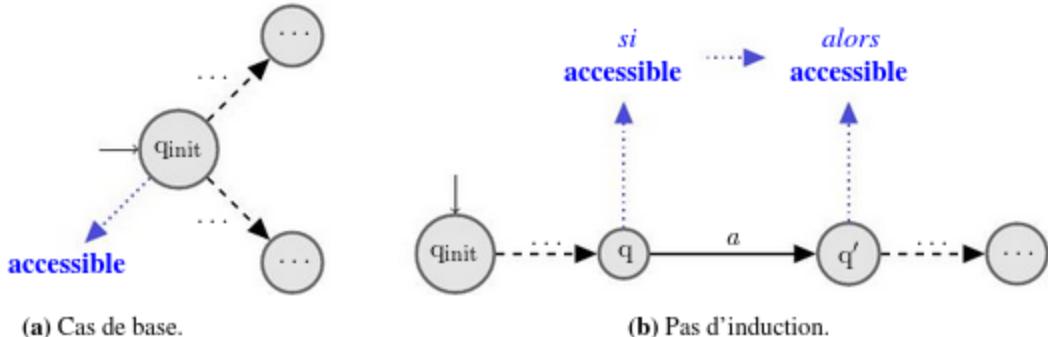


Figure 5.1 – Illustration de la définition inductive des états accessibles.

Définition 86 (Calcul des états accessibles) L'algorithme 4 (p. 95) calcule l'ensemble des états accessibles dans un AFED. L'algorithme utilise la fonction de transition des états pour calculer les états accessibles « de proche en proche » en ajoutant dans les états accessibles les états successeurs des états connus comme accessibles. Cet algorithme utilise et maintient trois ensembles d'états : Accessibles l'ensemble des états accessibles (résultat de l'algorithme), À_visiter qui est l'ensemble des états à visiter pour lesquels l'ensemble des successeurs n'ont pas encore été enregistrés, l'ensemble Déjà_visités qui est l'ensemble des états pour lesquels l'ensemble des successeurs ont été enregistrés. L'algorithme utilise aussi $\text{Succ}()$ qui permet d'obtenir les successeurs d'un état selon la fonction de transition de l'automate.

À l'initialisation, l'algorithme place l'état initial q_{init} dans l'ensemble des états accessibles et le marque comme à visiter. Par ailleurs, rien n'est visité. Ensuite, l'algorithme itère sur l'ensemble des états à visiter, qui est modifié par le corps de la boucle où des états sont ajoutés à mesure qu'ils sont découverts. Lors de chaque itération, un état q est sélectionné parmi les états à visiter et est supprimé de cet ensemble. L'état est ajouté à l'ensemble des états déjà visités. Dans l'ensemble des états accessibles, l'algorithme ajoute tous les successeurs de l'état q . De plus, l'algorithme ajoute les successeurs de q qui n'ont pas été visités dans l'ensemble des états à visiter (pour les prochaines itérations).

Concernant la terminaison, cet algorithme étant une adaptation de l'algorithme de parcours itératif des états de l'automate, les arguments sont similaires.

Remarque 21 L'algorithme 4 s'adapte facilement pour calculer les états accessibles à partir d'un état q_{depart} ou d'un ensemble d'états Départ de l'automate : en modifiant les lignes 2 et 3, en remplaçant $\{q_{\text{init}}\}$ par q_{depart} ou Départ.

Coaccessibilité

Rappelons qu'un état $q \in Q$ est coaccessible s'il existe un mot $u \in \Sigma^*$ tel que $\delta^*(q, u) \in F$. Pour calculer les états accessibles avec un algorithme itératif, il faut passer à une définition inductive de la notion d'accessibilité.

Algorithme 4 *états_accessiblees()* pour le calcul des états accessibles

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED*)

Sortie : $\text{Accessibles}(A)$ (* ensemble des états accessibles dans A par δ *)

- 1: **ensemble d'états** Accessibles, $\mathbb{A}_{\text{visiter}}$, Déjà_visités ;
- 2: Accessibles := $\{q_{\text{init}}\}$; (* cas de base *)
- 3: $\mathbb{A}_{\text{visiter}} := \{q_{\text{init}}\}$; (* parcours depuis l'état initial *)
- 4: $\text{Déjà_visités} := \emptyset$; (* initialement, rien n'est visité *)
- 5: **tant que** $\mathbb{A}_{\text{visiter}} \neq \emptyset$ faire (* tant qu'il reste un état à visiter *)
- 6: soit $q \in \mathbb{A}_{\text{visiter}}$; (* on choisit un état à visiter *)
- 7: $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{q\}$; (* l'état q n'est plus à visiter *)
- 8: $\text{Déjà_visités} := \text{Déjà_visités} \cup \{q\}$; (* l'état q est (presque) déjà visité *)
- 9: Accessibles := Accessibles $\cup \text{Succ}(q)$; (* on ajoute les états successeurs de q à l'ensemble global des états accessibles *)
- 10: $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup (\text{Succ}(q) \setminus \text{Déjà_visités})$; (* les nouveaux états à visiter sont ceux « découverts » *)
- 11: **fin tant que**
- 12: **retourner** Accessibles;

Définition 87 (Ensemble des états coaccessibles) L'ensemble des états coaccessibles dans A , noté $\text{Coaccessibles}(A)$, est défini inductivement par :

- l'ensemble F ;
- la fonction $q \mapsto \text{Pré}(q)$.

C'est-à-dire, le cas de base est que les états accepteurs sont coaccessibles et le pas d'induction est que si $q \in Q$ est coaccessible dans A , alors tous les prédécesseurs de q (les états de $\text{Pré}(q)$) sont coaccessibles dans A . La figure 5.2 illustre cette définition.

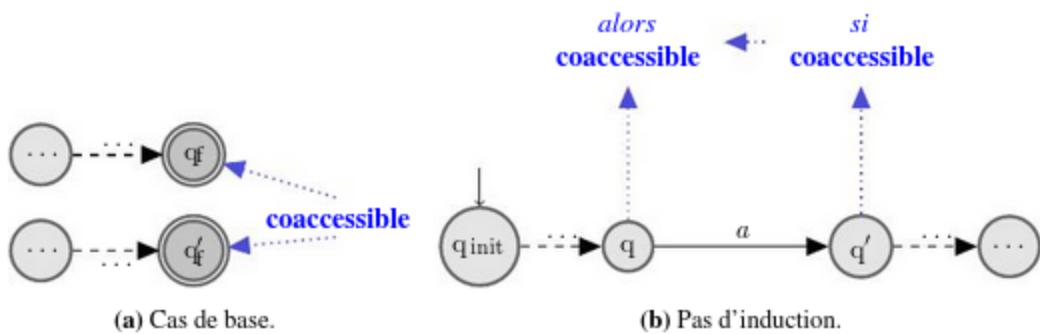


Figure 5.2 – Illustration de la définition inductive des états coaccessibles.

Définition 88 (Calcul des états coaccessibles) L'algorithme 5 (p. 96) calcule l'ensemble des états coaccessibles dans un AFED. Le principe est le même que celui de l'algorithme 4 (p. 95) pour le calcul des états accessibles. Les différences se situent dans l'initialisation de

l'algorithme qui marque tous les états accepteurs comme coaccessibles et utilise l'ensemble des états prédécesseurs lors des itérations.

Concernant la terminaison, cet algorithme étant une adaptation de l'algorithme de parcours itératif des états de l'automate, les arguments sont similaires.

Algorithme 5 `états_coaccessibles()` pour le calcul des états coaccessibles

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : $\text{Coaccessibles}(A)$ (* ensemble des états coaccessibles dans A par δ *)

- 1: **ensemble d'états** Coaccessibles, $\mathbb{A}_{\text{visiter}}$, $\text{Déjà}_\text{visités}$;
- 2: Coaccessibles := F ; (* cas de base *)
- 3: $\mathbb{A}_{\text{visiter}} := F$; (* on veut les états coaccessibles à tous les états accepteurs *)
- 4: $\text{Déjà}_\text{visités} := \emptyset$;
- 5: **tant que** $\mathbb{A}_{\text{visiter}} \neq \emptyset$ **faire** (* tant qu'il reste un état à visiter *)
 - 6: soit $q \in \mathbb{A}_{\text{visiter}}$; (* on choisit un état à visiter *)
 - 7: $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{q\}$; (* l'état q n'est plus à visiter *)
 - 8: $\text{Déjà}_\text{visités} := \text{Déjà}_\text{visités} \cup \{q\}$; (* l'état q est (presque) déjà visité *)
 - 9: Coaccessibles := Coaccessibles $\cup \text{Pré}(q)$;

(* on ajoute les états prédécesseurs de q à l'ensemble global des états accessibles *)
 - 10: $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup (\text{Pré}(q) \setminus \text{Déjà}_\text{visités})$;

(* les nouveaux états à visiter sont ceux « découverts » *)
- 11: **fin tant que**
- 12: **retourner** Coaccessibles;

Remarque 22 L'algorithme 5 peut facilement être modifié pour calculer les états coaccessibles à un état q_{depart} ou d'un ensemble d'états Départ de l'automate : en modifiant les lignes 2 et 3, en remplaçant $\{q_{\text{init}}\}$ par q_{depart} ou Départ.

Remarque 23 La coaccessibilité peut aussi se résoudre en réutilisant l'algorithme d'accessibilité. L'algorithme 6 (p. 96) réalise cela en construisant un automate intermédiaire E à partir de l'automate passé en entrée. On ajoute un nouvel état q_{new} à cet automate qui est l'état initial et qui sert pour le départ de la visite. Les transitions de l'automate intermédiaire sont d'une part les transitions de l'automate d'entrée que l'on inverse et d'autre part des transitions reliant q_{new} aux états accepteurs. Ensuite, on calcule les états accessibles dans ce nouvel automate et on retourne cet ensemble privé de l'état q_{new} . Notons que nous avons

Algorithme 6 `états_coaccessibles()` pour le calcul des états coaccessibles en utilisant l'accessibilité

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : ensemble des états coaccessibles dans A par δ

- 1: **ensemble de transitions** δ' ; (* fonction de transition de l'automate intermédiaire *)
- 2: $\delta' := \{(q', a, q) \in Q \times \Sigma \times Q \mid (q, a, q') \in \delta\} \cup (\{q_{\text{new}}\} \times \Sigma \times F)$;
- 3: **automate** $E := (Q \cup \{q_{\text{new}}\}, \Sigma, q_{\text{new}}, \delta', F)$;
- 4: **retourner** `états_accessibles(E) \ {q_{\text{new}}}`;

gardé arbitrairement le même ensemble d'états accepteurs ; cet ensemble n'influence pas le résultat du calcul.

5.2.4 Détection de cycles

Définition 89 (Cycle) Un cycle est une séquence (non vide) de transitions consécutives (l'état d'arrivée d'une transition est l'état de départ de la prochaine transition) telle que le l'état de départ de la première transition et l'état d'arrivée de la dernière transitions soient identiques.

Propriété 4 (Existence d'un cycle) Un automate possède un cycle s'il possède une transition dite arrière, c'est-à-dire une transition qui satisfait l'une des deux conditions suivantes :

- une transition d'un état sur lui-même,
- une transition d'un état vers l'un de ces ancêtres dans l'arbre produit par le parcours en profondeur.

Définition 90 (Détection de cycle) L'algorithme 7 (p. 98) détermine l'existence d'un cycle dans un AFED en se basant sur la propriété 4. Intuitivement, cet algorithme est une adaptation de l'algorithme de parcours en profondeur en utilisant une pile contenant des états liés par une transition dans l'automate. Lors de la visite d'un état, l'existence d'un cycle est ramenée à la présence d'un état successeur dans la pile.

L'algorithme utilise l'algorithme 8 (p. 98) qu'il appelle sur chaque état $q \in Q$ de l'automate. Ce dernier algorithme est récursif car il s'appelle sur les états successeurs de l'état sur lequel il est appelé. L'algorithme utilise les variables globales (aux deux algorithmes) Déjà_visités et Pile_d_appel. La variable Déjà_visités contient l'ensemble des états qui ont été déjà visités par l'algorithme, c'est-à-dire pour lesquels on a fait une détection de cycle partant de ces états. Cette variable permet notamment d'éviter de vérifier l'existence de cycle à partir d'un état de manière redondante. La variable Pile_d_appel est un ensemble d'états géré comme une pile. Il contient l'ensemble des états sur lesquels détection_cycle_état() a été appelé lors des appels précédents. La séquence d'états depuis le fond de pile jusqu'au sommet de pile correspond à la séquence des appels récursifs de détection_cycle_état() (l'état dans le fond de pile étant le premier état sur lequel détection_cycle_état() a été appelée). On note en particulier que détection_cycle_état() est appelé sur tous les successeurs de l'état courant et que l'état courant n'est retiré de la pile d'appel que quand les appels sur tous les successeurs sont terminés.

Concernant la terminaison, cet algorithme étant une adaptation de l'algorithme de parcours récursif des états de l'automate, les arguments sont similaires.

Remarque 24 L'algorithme 7 s'adapte facilement pour retourner tous les cycles.

Vocabulaire et utilisation

Un automate dont tous les états sont accessibles est dit *accessible*. Un automate dont tous les états sont coaccessibles est dit *coaccessible*. Un automate accessible et coaccessible est dit *émondé*. Intuitivement, dans un automate émondé, tous les états sont « utiles » à la reconnaissance des mots. Nous donnons quelques utilisation typique du calcul des états accessibles.

- on peut l'exprimer mathématiquement (*formellement*) ;
- elle a un certain nombre de paramètres que l'on souhaite pouvoir passer à un algorithme ou programme informatique ;
- sa réponse est oui ou non.

Définition 92 (Problème décidable et problème indécidable) *On peut distinguer deux types de problème de décision :*

- *Les problèmes décidables, pour lesquels on peut trouver un algorithme ou un programme qui sait répondre à la question (avec une mémoire et un temps non bornés) ; un tel algorithme est appelé procédure de décision.*
- *Les problèmes indécidables, pour lesquels on ne peut pas trouver un algorithme ou un programme qui sait répondre à la question (de manière générale).*

Nous considérons les deux problèmes de décision suivants :

Définition 93 (Problème du langage vide) *Le langage reconnu par A est-il vide ?*

Définition 94 (Problème de la finitude du langage) *Le langage reconnu par A est-il (de cardinalité) fini(e) ?*

Pour répondre à ces questions, nous allons utiliser dans les algorithmes de résolution de ces problèmes, appelés *procédures de décision*, les notions d'*accessibilité*, de *coaccessibilité* et de *cycles*. Ainsi, nous considérons deux problèmes de décision supplémentaires :

Définition 95 (Problème de l'accessibilité) *Un état $q \in Q$ est-il accessible dans A ?*

Définition 96 (Problème de la coaccessibilité) *Un état $q \in Q$ est-il coaccessible dans A ?*

Théorème 3 (Accessibilité dans les graphes finis) *Le problème de l'accessibilité est décidable (pour les AFED). L'algorithme 9 résout ce problème.*

Algorithme 9 Procédure de décision pour le problème de l'accessibilité

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Entrée : $q \in Q$ (* un état *)

Sortie : vrai si l'état q est accessible dans A , faux sinon

1: **ensemble d'états** Accessibles := $\text{états_accessibles}(A)$; (* calcul des états accessibles *)

2: **retourner** ($q \in \text{Accessibles}$);

Théorème 4 (Coaccessibilité dans les graphes finis) *Le problème de la coaccessibilité est décidable (pour les AFED). L'algorithme 10 (p. 100) résout ce problème.*

Algorithme 10 Procédure de décision pour le problème de la coaccessibilité

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Entrée: $q \in Q$ (* un état *)

Sortie : vrai si l'état q est coaccessible dans A , faux sinon

1: **ensemble d'états** coAccessibles := *états_coaccessibles*(*A*);

(* calcul des états coaccessibles *)

Théorème 5 (Décision du problème du langage vide) *Le problème du langage vide est décidable pour les automates (déterministes). L'algorithme 11 résout ce problème.*

Intuitivement, l'algorithme détermine s'il existe un état accepteur accessible dans l'automate.

Algorithme 11 Procédure de décision pour le problème du langage vide

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : vrai si le langage reconnu par A est vide, faux sinon

1: **ensemble d'états Accessibles** := $\text{états_accessibles}(A)$;

(* calcul des états accessibles *)

2: **retourner** ($\text{Accessibles} \cap F$) == \emptyset ;

Théorème 6 (Décision du problème de la finitude du langage) *Le problème de la finitude du langage est décidable pour les AFED. L'algorithme 12 résout ce problème.*

Intuitivement, l'idée utilisée dans cet algorithme est comme suit. Le langage accepté ne sera pas fini, si l'automate peut accepter « autant de mots qu'on veut ». Un AFED a un nombre fini d'états. Il faut pouvoir donc arriver dans un état accepteur par un nombre infini de chemins différents. Il faut donc l'existence d'un cycle. Ainsi, le langage reconnu par un automate $(Q, \Sigma, q_{\text{init}}, \delta, F)$ est infini si et seulement s'il existe un cycle accessible et coaccessible.

Algorithme 12 Procédure de décision pour le problème du langage infini

Entrée: $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : vrai si le langage reconnu par A est infini, faux sinon

2: **automate** $E := (\text{EtatsEmonde}, \Sigma, q_{\text{init}}, \delta \cap \text{EtatsEmonde} \times \Sigma \times \text{EtatsEmonde}, F \cap \text{EtatsEmonde})$; (* version émondée de A *)

3: **retourner** *détection_cycle*(*E*);

5.3 Exercices

5.3.1 Complétion, complémentation et produit

Exercice 45 (♠♦) — Algorithmes pour compléter

Considérons un automate sur un alphabet Σ .

- À partir de la définition de l'automate complété, définition 78 (p. 72), donner un algorithme complétant un automate par rapport à un alphabet Σ' tel que $\Sigma \subseteq \Sigma'$.
- Si nécessaire, adapter l'algorithme donné à la question précédente pour qu'il fonctionne avec un alphabet Σ' quelconque.

Exercice 46 (♠♦) — Algorithmes pour déterminer la complétude

Considérons deux alphabets Σ et Σ' tels que $\Sigma' \subseteq \Sigma$. Nous rappelons qu'un automate sur un alphabet Σ est dit complet si sa fonction de transition est définie pour chaque symbole de Σ en chaque état.

- Donner un algorithme qui détermine si un automate sur un alphabet Σ est complet.
- Nous disons qu'un automate sur l'alphabet Σ est complet par rapport à l'alphabet Σ' si sa fonction de transition est définie en chaque état sur chaque symbole de Σ' . Donner un algorithme qui détermine si un automate sur l'alphabet Σ est complet sur l'alphabet Σ' .

Exercice 47 (♠♦) — Algorithmes pour complémenter

Considérons la définition de l'automate complémentaire, définition 79 (p. 72).

- Donner un algorithme prenant en entrée un automate et produisant un automate reconnaissant le complémentaire du langage reconnu par l'automate passé en paramètre.

Exercice 48 (♠♦) — Algorithme pour calculer l'automate produit

Considérons un alphabet Σ .

- À partir de la définition de l'automate produit, définition 80 (p. 73), donner un algorithme produisant un automate qui reconnaît l'intersection des langages des automates passés en paramètres. La construction de l'ensemble d'états de l'automate produit doit se faire « à la volée », c'est-à-dire en construisant les états de l'automate produit de manière paresseuse en parcourant simultanément les ensembles d'états des automates de départ.
- Pour rappel, selon la définition d'automate produit, si les automates passés en paramètres ont pour ensemble d'états Q_1 et Q_2 , alors l'automate produit a pour ensemble d'états $Q_1 \times Q_2$ (avec $|Q_1 \times Q_2| = |Q_1| \times |Q_2|$). Donner des exemples d'automates tels que votre algorithme produise un automate dont le cardinal soit strictement inférieur à $|Q_1 \times Q_2|$.
- Donner une condition suffisante pour que l'algorithme produise un automate dont le cardinal soit strictement inférieur à $|Q_1 \times Q_2|$.

5.3.2 Émonder un automate

Exercice 49 (♠♠) — Déterminer si un automate est émondé

1. Donner un algorithme qui détermine si un automate est accessible.
2. Donner un algorithme qui détermine si un automate est coaccessible.
3. Donner un algorithme qui détermine si un automate est émondé. Cet algorithme pourra réutiliser les algorithmes trouvés aux questions précédentes.

Exercice 50 (♠♠) — Émonder un automate

Soient Σ un alphabet. Donner des algorithmes qui résolvent les problèmes suivants.

1. Supprimer les états non accessibles et les transitions impliquant ces états (c'est-à-dire une transition telle que l'état de départ ou l'état d'arrivée soit un état supprimé).
2. Supprimer les états non coaccessibles et les transitions reliées à ces états.
3. Émonder un automate.

Exercice 51 (♠♠♣) — Automate émondé

1. Montrer qu'à tout automate, on peut associer un automate émondé qui reconnaît le même langage.

5.3.3 Inclusion et égalité des langages reconnus par des automates

Exercice 52 (♠♠♣) — Inclusion entre langages reconnus

Considérons deux langages à états L et L' et les automates A_L et $A_{L'}$ reconnaissant L et L' , respectivement.

1. En utilisant les opérateurs d'intersection et de complémentation entre langages, écrire une relation entre langages équivalente à $L \subseteq L'$.
2. Déduire de la question précédente un algorithme permettant de déterminer si $L \subseteq L'$.
3. Déduire de la question précédente un algorithme permettant de déterminer si $L = L'$.

Exercice 53 (♠♠♣♣) — Égalité modulo un renommage des états

Étant donnés deux AFED, nous nous intéressons au problème de déterminer si ces automates sont égaux modulo un renommage des états. Ceci est une condition suffisante pour que ces deux automates reconnaissent le même langage.

Deux AFED $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ sont égaux modulo un renommage des états s'il existe une fonction bijective de Q^A vers Q^B telle que $Q^B = f(Q^A)$, $q_{\text{init}}^B = f(q_{\text{init}}^A)$, $\delta^B = f(\delta^A)$ et $F^B = f(F^A)$ où f est étendue à un ensemble d'états $E \subseteq Q^A$ et à une fonction de transition sur les états de Q^A et les symboles de Σ comme suit : $f(E) = \{f(q) \mid q \in E\}$ et $f(\delta^A) = \{(f(q), s, f(q')) \mid q, q' \in Q^A \wedge s \in \Sigma\}$.

1. Écrire un algorithme qui détermine si deux AFED sont égaux modulo un renommage des états.

5.3.4 Autour de la notion de préfixe

Exercice 54 (♠♠♠) — Langages des préfixes

Soit L un langage sur un alphabet Σ . Pour rappel, l'ensemble des préfixes (des mots) de L est l'ensemble :

$$\text{Pref}(L) = \{u \in \Sigma^* \mid \exists u' \in \Sigma^*. u \cdot u' \in L\}.$$

Supposons que L soit un langage à états.

1. Montrer que $\text{Pref}(L)$ est également un langage à états.

Exercice 55 (♠♠♠) — Des automates qui reconnaissent les langages préfixe-clos

Nous souhaitons montrer que le problème de déterminer si le langage reconnu par un automate est préfixe-clos est décidable.

1. Donner des exemples et des contre-exemples d'AFED qui reconnaissent des langages préfixe-clos.
2. Caractériser par une condition nécessaire et suffisante les AFEDS qui reconnaissent les langages préfixe-clos.
3. Donner un algorithme qui permet de décider si un langage défini par un AFED est préfixe-clos.
4. Tester votre algorithme sur les automates de la première question.

Exercice 56 (♠♠♠) — Plus grand sous-ensemble préfixe-clos d'un langage accepté

Nous considérons l'algorithme trouvé dans l'exercice 55.

1. En s'inspirant de cet algorithme, donner un algorithme qui transforme un automate de telle manière à ce que le langage reconnu par l'automate résultat soit le plus grand sous-ensemble préfixe-clos du langage reconnu par l'automate initial.

5.4 Indications pour résoudre les exercices

5.4.1 Compléter un automate

Indications pour l'exercice 45 (p. 101)

1. Pour compléter l'automate par rapport à un alphabet Σ' qui est un sur-ensemble de l'alphabet de l'automate Σ , il faut pour chaque état de l'automate, pour chaque symbole, vérifier qu'il y ait une transition définie dans la fonction de transition, et si cette transition n'existe pas l'ajouter.

2. Pour chaque symbole de Σ' , il suffit de compléter l'automate suivant le même principe que pour la première question. Il faut de plus traiter les symboles de $\Sigma \setminus \Sigma'$. Nous pouvons par exemple faire le choix de les supprimer pour que cette opération de complétiōn réalise également un changement d'alphabet de l'automate.

Indications pour l'exercice 46 (p. 101)

1. Il faut suivre le même principe que pour l'exercice 45. L'algorithme doit retourner faux dès qu'il trouve un état et un symbole pour lequel la fonction de transition n'est pas définie. Lorsque tous les états et tous les symboles ont été passés en revue, l'algorithme retourne vrai.

Indications pour l'exercice 47 (p. 101)

1. Utiliser la définition de l'automate complémentaire, définition 79 (p. 72). Réutiliser l'algorithme trouvé dans l'exercice 45.

Indications pour l'exercice 48 (p. 101)

1. Il y a plusieurs solutions possibles. Premièrement, il est possible de faire un parcours de l'automate produit en utilisant la définition de la fonction de transition pour découvrir de nouveaux états à visiter. Deuxièmement, il est possible de faire une exploration des états en utilisant la définition de l'automate produit (et de compléter sa fonction de transition au fur et à mesure). Pour cela, on peut faire une itération non bornée où on utilise deux ensembles de variables : un ensemble contenant les états découverts jusqu'à l'itération courante, l'autre contenant les états découverts jusqu'à l'itération précédente. Avec ces deux ensembles d'états, il est possible de déterminer lorsqu'on ne découvre plus d'états.
2. Appliquer simplement la construction de l'automate produit.
3. Dans l'automate $A \times B$, tous les états du produit cartésien entre Q^A et Q^B sont présents. Dans l'automate produit par l'algorithme, seuls les états accessibles sont présents. Déterminer à quelle condition un état dans $Q^A \times Q^B$ peut ne pas être accessible.

5.4.2 Émonder un automate

Indications pour l'exercice 49 (p. 102)

1. Utiliser l'algorithme de calcul des états accessibles, algorithme 4 (p. 95).
2. Utiliser l'algorithme de calcul des états coaccessibles, algorithme 5 (p. 96).
3. Utiliser les algorithmes trouvés dans les deux questions précédentes.

Indications pour l'exercice 50 (p. 102)

1. *Garder uniquement les états accessibles et supprimer les transitions impliquant un état non accessible.*

Indications pour l'exercice 51 (p. 102)

1. *Utiliser l'automate produit par l'algorithme trouvé dans l'exercice 50. Ensuite, pour démontrer que l'automate de départ et l'automate émondé reconnaissent les mêmes langages, démontrer que si un mot est accepté par l'un des deux automates, alors il est accepté par l'autre automate.*

5.4.3 Inclusion et égalité des langages reconnus par des automates**Indications pour l'exercice 52 (p. 102)**

1. *Utiliser le complémentaire de l'un des langages.*
2. *Associer aux opérations sur les langages, les opérations sur les automates.*
3. *Associer aux opérations sur les langages, les opérations sur les automates.*

Indications pour l'exercice 53 (p. 102)

1. *Une idée est de faire un parcours simultané des deux automates en construisant la fonction de renommage à la volée. Nous représentons la fonction de renommage par un ensemble de couples : le premier élément du couple se réécrit en le second élément du couple selon la fonction de renommage. Initialement, l'association se fait entre les deux états initiaux. Ensuite, on parcourt les états en ayant une association déjà construite entre les états et en ayant un ensemble d'états à visiter qui sont des couples représentant les associations faites jusqu'à présent. À chaque étape du parcours, on sélectionne un état à visiter. Sur chaque symbole, examiner les successeurs, et distinguer les différentes situations.*

5.4.4 Autour de la notion de préfixe**Indications pour l'exercice 54 (p. 103)**

1. *À partir de l'automate reconnaissant L , construire un automate reconnaissant $\text{Pref}(L)$.*

Indications pour l'exercice 55 (p. 103)

2. *Dans un langage préfixe-clos, si un mot est accepté alors tous ses préfixes le sont également. Traduire cette condition s'exprime sur la fonction de transition de l'automate.*

3. Si il existe une transition depuis un état accessible non accepteur vers un état accessible accepteur, alors le langage n'est pas préfixe clos. On peut réutiliser l'automate émondé (exercice 49). On peut tester l'existence d'une telle transition ou tester l'existence d'un état non accepteur accessible et coaccessible.

Indications pour l'exercice 56 (p. 103)

- En s'inspirant de l'algorithme trouvé dans l'exercice 55, identifier les états faisant que le langage reconnu n'est pas préfixe-clos. Supprimer ces états et les transitions reliées à ces états.

5.5 Solutions des exercices

5.5.1 Compléter un automate

Solution de l'exercice 45 (page 101)

- Un algorithme itératif pour compléter un AFED donné en paramètre par rapport à un alphabet Σ' tel que $\Sigma' \supseteq \Sigma$, où Σ est l'alphabet de l'automate est donné dans l'algorithme 13. Cet algorithme réalise un parcours des états de l'automate et vérifie que cet état est la source d'une transition sur chaque symbole de l'alphabet.

Algorithme 13 *compléter()* pour compléter un automate par rapport à un alphabet Σ' tel que $\Sigma' \supseteq \Sigma$, où Σ est l'alphabet de l'automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Entrée : $\Sigma' \supseteq \Sigma$ (* un alphabet sur-ensemble de l'alphabet de l'automate *)
Sortie : $C(A)$ (* l'AFED complet construit à partir de A *)

- 1: **ensemble de transitions** δ_{tmp} ; (* transitions ajoutées pour compléter l'automate *)
- 2: **état** q_p ; (* état puits $q_p \notin Q$ *)
- 3: $\delta_{\text{tmp}} := \emptyset$;
- 4: **pour** chaque état $q \in Q$ **faire**
- 5: **pour** chaque symbole $s \in \Sigma'$ **faire**
- 6: **si** $\nexists q' \in Q$. $\delta(q, a) = q'$ **alors**
 (* s'il n'y a pas de transition depuis l'état q sur le symbole s *)
- 7: $\delta_{\text{tmp}} := \delta_{\text{tmp}} \cup \{(q, s, q_p)\}$;
 (* on ajoute une transition depuis l'état q vers l'état q_p sur le symbole s *)
- 8: **fin si**
- 9: **fin pour**
- 10: **fin pour**
- 11: **retourner** $(Q \cup \{q_p\}, \Sigma, q_{\text{init}}, \delta \cup \delta_{\text{tmp}}, F)$;

- Un algorithme pour compléter un automate par rapport à un alphabet Σ' quelconque est donné dans l'algorithme 14. L'énoncé ne précisait pas comment traiter les transitions sur les symboles dans $\Sigma \setminus \Sigma'$, nous faisons le choix de supprimer ces transitions. Cet

algorithme traite séparément les transition sur les symboles dans $\Sigma \setminus \Sigma'$ et complète les transitions dans Σ' . L'algorithme supprime les transitions

Algorithme 14 *compléter()* pour compléter un automate par rapport à un alphabet Σ' quelconque

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Entrée : Σ' (* un alphabet quelconque *)
Sortie : L'AFED complet construit à partir de A

- 1: **ensemble de** transitions δ_{tmp} ; (* transitions ajoutées pour compléter l'automate *)
- 2: **état** q_p ; (* on crée un nouvel état $q_p \notin Q$ *)
- 3: $\delta_{\text{tmp}} := \emptyset$;
- 4: **pour** chaque état $q \in Q$ **faire** (* on complète l'automate sur les symboles de Σ' *)
- 5: **pour** chaque symbole $s \in \Sigma'$ **faire**
- 6: **si** $\nexists q' \in Q$. $\delta(q, s) = q'$ **alors**
 (* s'il n'y a pas de transition depuis l'état q sur le symbole s *)
- 7: $\delta_{\text{tmp}} := \delta_{\text{tmp}} \cup \{(q, s, q_p)\}$;
 (* ajout d'une transition depuis q vers q_p sur s *)
- 8: **fin si**
- 9: **fin pour**
- 10: **pour** chaque symbole $s \in \Sigma \setminus \Sigma'$ **faire**
- 11: **si** $\exists q' \in Q$. $\delta(q, s) = q'$ **alors**
- 12: $\delta := \delta \setminus \{(q, s, q')\}$;
 (* suppression des transitions sur les symboles de Σ qui ne sont pas de Σ' *)
- 13: **fin si**
- 14: **fin pour**
- 15: **fin pour**
- 16: **retourner** $(Q \cup \{q_p\}, \Sigma', q_{\text{init}}, \delta \cup \delta_{\text{tmp}}, F)$;

Solution de l'exercice 46 (page 101)

1. Un algorithme qui détermine si un automate passé en paramètre est complet est donné dans l'algorithme 15.

Algorithme 15 *est_complet()* pour déterminer si un automate est complet

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : vrai si A est complet, faux sinon

- 1: **pour** chaque état $q \in Q$ **faire**
- 2: **pour** chaque symbole $s \in \Sigma$ **faire**
- 3: **si** $\nexists q' \in Q$. $\delta(q, s) = q'$ **alors**
 (* s'il n'y a pas de transition depuis q sur s *)
- 4: **retourner** faux;
- 5: **fin si**
- 6: **fin pour**
- 7: **fin pour**
- 8: **retourner** vrai;

2. Pour obtenir l'algorithme demandé, il suffit de changer la ligne 2 dans l'algorithme 15 en remplaçant Σ par Σ' .

Solution de l'exercice 47 (page 101)

1. Un algorithme produisant un automate reconnaissant le langage complémentaire du langage reconnu par un automate passé en paramètre est donné dans l'algorithme 16. Cet algorithme utilise l'algorithme *compléter()* (algorithme 13) qui complète un automate passé en paramètre. Notons que la procédure de complétion modifie uniquement l'ensemble des états et les transitions de l'automate auquel elle s'applique.

Algorithme 16 *compléter()* pour compléter un automate passé en paramètre

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : A^c (* l'automate complémentaire de A *)

- 1: $(Q', \Sigma, q_{\text{init}}, \delta', F) := \text{compléter}(A);$
- 2: **retourner** $(Q', \Sigma, q_{\text{init}}, \delta', Q' \setminus F);$
(* inversion états accepteurs/non accepteurs de l'automate complet *)

Remarque 25 Concernant l'implémentation de l'algorithme 16, si le fait qu'un état soit accepteur est indiqué par une propriété de l'état, pour inverser état accepteurs et non accepteurs, il faudrait itérer sur Q' en inversant la propriété pour chaque état.

Solution de l'exercice 48 (page 101)

1. Pour réaliser le produit, nous proposons l'algorithme 17. L'ensemble Q contient les états de l'automate produit en construction. L'algorithme explore les états de l'automate produit en utilisant la fonction de transition à partir des états découverts. Chaque itération utilise les états découverts à l'itération précédente et découvre des états qui sont explorés à l'itération suivante. Pour cela, l'algorithme utilise deux ensembles d'états :

- l'ensemble Q_{new} qui contient les états créés pendant l'itération courante et qui sont à explorer à l'itération suivante ;
- l'ensemble Q_{pre} qui contient la valeur de Q_{new} à l'itération précédente.

À l'initialisation, l'ensemble d'états de l'automate produit Q est $\{(q_{\text{init}}^A, q_{\text{init}}^B)\}$, c'est-à-dire le singleton contenant l'état initial qui est le couple formé par les deux états initiaux. Q_{pre} est initialisé à \emptyset et Q_{new} à $\{(q_{\text{init}}^A, q_{\text{init}}^B)\}$. L'algorithme itère tant que Q_{new} est non vide. Lors de chaque itération, l'algorithme commence à mettre à jour les variables Q_{pre} et Q_{new} . La variable Q_{pre} prend la valeur de Q_{new} qui va contenir les états à explorer et sera utilisée en lecture (alors que la variable Q_{new} sera modifiée durant l'itération). La variable Q_{new} est initialisée à \emptyset car l'algorithme va y stocker les états à explorer lors de la prochaine itération. Ensuite, pour chaque état (q^A, q^B) à explorer (stocké dans Q_{pre}), pour chaque symbole de l'alphabet, l'algorithme calcule l'état successeur selon la fonction de transition suivant la définition de l'automate produit (à partir des états q^A et q^B dans les automates A et B respectivement). La transition depuis l'état présentement exploré vers l'état découvert est ajoutée

à la fonction de transition de l'automate. À la fin de l'exploration de Q_{pre} , les états découverts stockés dans Q_{new} sont ajoutés à Q .

Algorithme 17 *produit()* pour calculer le produit de deux automates

Entrée : $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ (* deux AFED *)
Sortie : $A \times B$, calculé « à la volée » (* l'automate produit de A et B *)

1: **ensemble d'états** $Q := \{(q_{\text{init}}^A, q_{\text{init}}^B)\}$, $Q_{\text{pre}} := \emptyset$, $Q_{\text{new}} := \{(q_{\text{init}}^A, q_{\text{init}}^B)\}$;
 2: **ensemble de transitions** $\delta := \emptyset$;
 3: **état** q ;
 4: **tant que** $Q_{\text{new}} \neq \emptyset$ **faire**
 5: $Q_{\text{pre}} := Q_{\text{new}}$; (* sauvegarde de Q_{new} dans Q_{pre} *)
 6: $Q_{\text{new}} := \emptyset$; (* mise à zéro de Q_{new} pour préparer à la prochaine itération *)
 7: **pour** chaque $(q^A, q^B) \in Q_{\text{pre}}$ **faire**
 8: **pour** chaque $s \in \Sigma$ **faire**
 9: **si** $(\exists q^{A'} \in Q^A . q^{A'} = \delta^A(q^A, s)) \wedge (\exists q^{B'} \in Q^B . q^{B'} = \delta^B(q^B, s))$ **alors**
 (* s'il existe une transition sur le symbole s depuis les états q^A et q^B *)
 10: $q := (q^{A'}, q^{B'})$;
 11: $\delta := \delta \cup \{((q^A, q^B), s, q)\}$; (* ajout de la transition vers l'état découvert *)
 12: **si** $q \notin Q$ **alors**
 13: $Q_{\text{new}} := Q_{\text{new}} \cup \{q\}$; (* ajout de l'état découvert à Q_{new} *)
 14: **fin si**
 15: **fin si**
 16: **fin pour**
 17: **fin pour**
 18: $Q := Q \cup Q_{\text{new}}$; (* ajout des états découverts lors de cette itération à l'ensemble des états du produit. *)
 19: **fin tant que**
 20: **retourner** $(Q, \Sigma, (q_{\text{init}}^A, q_{\text{init}}^B), \delta, (F^A \times F^B) \cap Q)$;

2. Nous proposons les automates représentés dans les figures 5.3a et 5.3b sur l'alphabet $\{a, b\}$. L'automate produit est représenté dans la figure 5.3c.

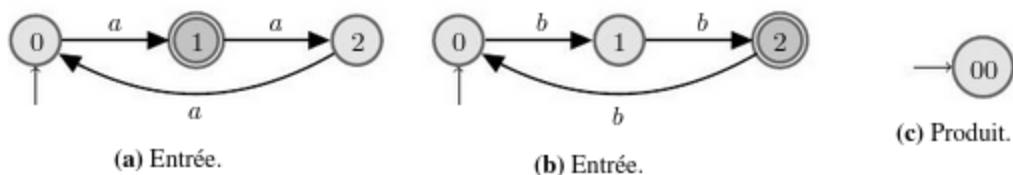


Figure 5.3 – Exemples d'automates utilisés dans l'exercice 48 (p. 101) pour lesquels l'automate produit (1 état) a strictement moins d'états que le produit cartésien des ensembles d'états des automates utilisés pour le calculer (9 états).

3. Soient $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ les deux automates utilisés en entrée de l'algorithme et $A \times B = (Q, \Sigma, q_{\text{init}}, \delta, F)$ l'automate produit selon la définition 80. Nous remarquons que l'algorithme 17 construit 1) un automate dont tous les états sont accessibles et que 2) les états de cet automate sont obtenus à partir des états accessibles des automates A et B utilisés en entrée.

Nous donnons quelques exemples de conditions suffisantes.

1. $\text{Accessibles}(A) \neq Q^A$;
2. $\text{Accessibles}(B) \neq Q^B$;
3. $\exists (q^A, q^B) \in \text{Accessibles}(A \times B). \exists s \in \Sigma. \exists q^{A'} \in Q^A. \nexists q^{B'} \in Q^B. \left((q^A, s, q^{A'}) \in \delta^A \wedge (q^B, s, q^{B'}) \in \delta^B \right)$;
4. $\exists (q^A, q^B) \in \text{Accessibles}(A \times B). \exists s \in \Sigma. \nexists q^{A'} \in Q^A. \exists q^{B'} \in Q^B. \left((q^A, s, q^{A'}) \in \delta^A \wedge (q^B, s, q^{B'}) \in \delta^B \right)$;
5. $\text{Accessibles}(A \times B) \neq Q$.

La première et la seconde condition expriment que les automates A et B ne sont pas accessibles, respectivement. Lorsqu'un des automates n'est pas accessible, par définition un de ses états n'est pas accessible, et cet état ne participera pas à la construction de l'automate produit. La troisième condition exprime qu'il existe un état accessible du produit (q^A, q^B) tel qu'il existe une transition à partir de l'état q^A sur le symbole s dans A et qu'il n'existe pas de transition sur ce symbole à partir de l'état q^B dans B . En conséquence, dans l'automate produit, il n'y aura pas de successeur à l'état (q^A, q^B) sur le symbole s . La quatrième condition exprime la condition symétrique de la troisième où l'existence de la transition est sur B . La cinquième condition exprime que l'automate produit $A \times B$ n'est pas accessible. Ainsi l'automate $A \times B$ possède un état qui ne sera pas construit par l'algorithme 17.

5.5.2 Émonder un automate

Solution de l'exercice 49 (page 102)

1. L'algorithme est donné dans l'algorithme 18.
2. L'algorithme est donné dans l'algorithme 19.
3. Nous réutilisons les deux algorithmes trouvés aux deux questions précédentes. La solution la plus simple étant de vérifier que les tous les états sont accessibles et coaccessibles. L'algorithme est donné dans l'algorithme 20.

Algorithme 18 *est_accessible()* pour déterminer si un automate est accessible

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : vrai si A est accessible, faux sinon

- 1: Accessibles := *états_accessible(A)*;
 - 2: **retourner** Accessibles == Q ;
-

Algorithme 19 *est_coaccessible()* pour déterminer si un automate est coaccessible

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : vrai si A est coaccessible, faux sinon
 1: `coAccessibles := états_coaccessibles(A);`
 2: **retourner** `coAccessibles == Q` ;

Algorithme 20 *est_émondé()* pour déterminer si un automate est émondé

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : vrai si A est émondé, faux sinon
 1: **retourner** `est_accessible(A) \& est_coaccessible(A)`;

Solution de l'exercice 50 (page 102)

1. Nous proposons l'algorithme 21 qui est une solution simple réutilisant l'algorithme 4 pour calcul des états accessibles.

Algorithme 21 *rendre_accessible()* pour produire la version accessible d'un automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)
Sortie : l'automate A où tous les états non accessibles et toutes les transitions impliquant ces états sont supprimés
 1: `Accessibles := états_accessible(A);`
 2: **pour** chaque $(q_{\text{start}}, s, q_{\text{end}}) \in \delta$ **faire**
 3: **si** $q_{\text{start}} \notin \text{Accessibles}$ ou $q_{\text{end}} \notin \text{Accessibles}$ **alors**
 4: $\delta := \delta \setminus \{(q_{\text{start}}, s, q_{\text{end}})\};$
 5: **fin si**
 6: **fin pour**
 7: **retourner** $(Q \cap \text{Accessibles}, \Sigma, q_{\text{init}}, \delta, F \cap \text{Accessibles})$;

2. L'algorithme est le même que celui de la question précédente à l'exception qu'à la première ligne, on appelle l'algorithme de calcul des états coaccessibles. Nous nommons cet algorithme *rendre_coaccessible()*, qui a la même « signature » que l'algorithme de la question précédente.
3. Émonder un automate consiste à supprimer les états accessibles et les états non coaccessibles. Nous nettoyons également la fonction de transition en supprimant les transitions impliquant les état non accessibles ou non coaccessibles. Ainsi, nous pouvons réutiliser directement les deux algorithmes trouvés aux deux questions précédentes pour obtenir l'algorithme 22.

Solution de l'exercice 51 (page 102)

1. Soit A un automate et $E = \text{émonder}(A)$ sa version émondée obtenue avec l'algorithme 22. Pour démontrer que A et E reconnaissent le même langage, nous montrons que si un mot est accepté par l'un des deux automates, alors il est accepté par l'autre automate.

Algorithme 22 émonder() pour produire la version émondée d'un automate

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)**Sortie :** l'automate A où tous les états non accessibles et toutes les transitions impliquant ces états sont supprimés1: **retourner** *rendre_accessible*(*rendre_coaccessible*(A));

- $\mathcal{L}_{\text{auto}}(A) \subseteq \mathcal{L}_{\text{auto}}(E)$. Soit u un mot accepté par A et considérons l'exécution de ce mot (qui termine dans un état accepteur). Par définition, les états apparaissant dans l'exécution de u sont accessibles. Ces états sont également coaccessibles ils permettent d'atteindre un état accepteur (celui atteint par l'exécution de u). Par ailleurs, émonder un automate, ne modifie ni les états ni les transitions ni les états accepteurs de l'automate. Ainsi, la même exécution de u existe sur E .
- $\mathcal{L}_{\text{auto}}(E) \subseteq \mathcal{L}_{\text{auto}}(A)$. Pour cette direction, il suffit de remarquer que la version émondée d'un automate en est un sous-automate. Ainsi, en utilisant le résultat de l'exercice 33 (p. 53), première question, nous obtenons le résultat recherché.

5.5.3 Inclusion et égalité des langages reconnus par des automates**Solution de l'exercice 52 (page 102)**

1. La condition équivalente à $L \subseteq L'$ est : $L \cap \overline{L'} = \emptyset$.
2. En réutilisant les algorithmes pour obtenir l'automate complémentaire et l'automate produit de deux automates, nous obtenons l'algorithme 23.

Algorithme 23 inclusion(A, B) pour tester l'inclusion des langages entre deux automates

Entrée : $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ (* deux AFED *)**Sortie :** vrai si $\mathcal{L}_{\text{auto}}(A) \subseteq \mathcal{L}_{\text{auto}}(B)$, faux sinon

- 1: $B := \text{complémenter}(B)$
- 2: AFED $C := \text{produit}(A, B)$
- 3: **retourner** *est_vide*(C);

3. Nous réutilisons l'algorithme de la question précédente et le fait que, pour deux langages (ou ensembles) L et L' , $L = L'$ si et seulement si $L \subseteq L'$ et $L' \subseteq L$. Nous obtenons l'algorithme 24.

Algorithme 24 égalité(A, B) pour tester l'égalité des langages entre deux automates

Entrée : $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ (* deux AFED *)**Sortie :** vrai si $\mathcal{L}_{\text{auto}}(A) = \mathcal{L}_{\text{auto}}(B)$, faux sinon1: **retourner** (*inclusion*(A, B) \wedge *inclusion*(B, A));

Solution de l'exercice 53 (page 102)

1. L'idée de l'algorithme est de construire une association entre états de Q^A et états de Q^B et d'utiliser les fonctions de transitions pour réaliser cette construction. La construction de cette fonction se fait en réalisant un parcours des états de l'automate, en commençant par associer l'état initial du premier automate avec l'état initial du second automate. Lors de la construction de cette association, nous réalisons deux opérations : d'abord, nous complétons la définition de l'association entre états et vérifions la compatibilité de l'association construite jusqu'à présent avec les transitions de l'automate. Comme l'automate est déterministe, lors de la visite d'un état, les nouvelles associations sont déterminées de manière unique et l'incompatibilité avec une seule transition nous donne l'incompatibilité de l'automate global.

L'algorithme 25 se base sur ce raisonnement et vérifie l'existence de l'association. L'algorithme parcourt simultanément les deux automates et utilise l'ensemble `Déjà_visités` pour stocker les associations déjà faites et vérifiées et l'ensemble `À_visiter` pour stocker l'ensemble des associations à vérifier. Pour chaque association existante et vérifiée (q^A, q^B) , les vérifications sont faites pour chaque symbole s de l'alphabet. Si aucun des états n'a de successeur par s , alors on ne fait rien. Si exactement l'un des deux états a un successeur par s , alors l'algorithme termine et retourne faux car il ne peut exister d'association valide. Sinon, nous considérons les successeurs q_s^A et q_s^B de q^A et q^B par s . Nous vérifions si le couple d'états (q_s^A, q_s^B) n'a pas été déjà vérifié ou déjà marqué comme à vérifier. Si l'un des deux états (q_s^A, q_s^B) a déjà été associé à un autre état (soit dans l'ensemble des états déjà vérifiés ou à vérifier), alors l'algorithme retourne faux. Sinon le couple (q_s^A, q_s^B) est placé dans les états à vérifier.

5.5.4 Autour de la notion de préfixe**Solution de l'exercice 54 (page 103)**

1. Soit $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ un automate reconnaissant L . L'idée est que l'automate résultat A_{pref} doit accepter un mot u si et seulement s'il existe un mot u' qui est une extension de u qui est accepté par A . Ainsi, comme en prenant le même ensemble d'états et la même fonction de transition, les exécutions des mots seront les mêmes sur A et A_{pref} . Un état de A_{pref} doit être marqué comme accepteur si et seulement si cet état est coaccessible dans A , c'est-à-dire que l'on peut atteindre un état accepteur de A à partir de cet état. Ceci correspond à l'équivalence suivante $u \in \mathcal{L}_{\text{auto}}(A_{\text{pref}})$ si et seulement si $\exists u'. u \cdot u' \in \mathcal{L}_{\text{auto}}(A)$. Nous en déduisons que l'automate A_{pref} est $(Q, \Sigma, q_{\text{init}}, \delta, \text{états_coaccessibles}(F))$ (noter que $F \subseteq \text{états_coaccessibles}(F)$).

Solution de l'exercice 55 (page 103)

- Des exemples d'automate reconnaissant des langages préfixe-clos et des automates reconnaissant des langages non préfixe-clos sont donnés dans la figure 5.4.
 - Pour les langages préfixe-clos, nous avons les automates suivants :
 - L'AFED dans la figure 5.4a reconnaît le langage sur un alphabet $\Sigma \supseteq \{a\}$ des mots contenant que des occurrences du symbole a .
 - L'AFED dans la figure 5.4b reconnaît le langage vide.

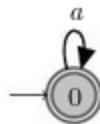
Algorithme 25 *existence_fonction_renommage()* pour le parcours à partir des états

Entrée : $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$ et $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ (* deux AFED *)

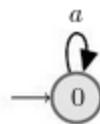
Sortie : vrai s'il existe une fonction de renommage des états entre A et B , faux sinon

```

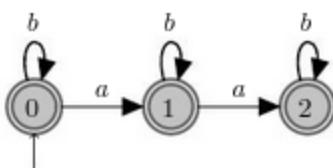
1: ensemble d'états  $\mathbb{A}_{\text{visiter}} := \{(q_{\text{init}}^A, q_{\text{init}}^B)\}$ ;
2: ensemble d'états  $\text{Déjà\_visités} := \emptyset$ ; (* initialement, rien n'est visité *)
3: tant que  $\mathbb{A}_{\text{visiter}} \neq \emptyset$  faire
4:   soit  $(q^A, q^B) \in \mathbb{A}_{\text{visiter}}$ ; (* prendre un couple d'états à visiter *)
5:    $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{(q^A, q^B)\}$ ; (* le couple d'états  $(q^A, q^B)$  n'est plus à visiter *)
6:    $\text{Déjà\_visités} := \text{Déjà\_visités} \cup \{(q^A, q^B)\}$ ; (* le couple d'états  $(q, q')$  vient d'être visité *)
7:   pour  $s \in \Sigma$  faire
8:     si  $(\nexists q_s^A \in Q^A \cdot (q^A, s, q_s^A) \in \delta^A) \wedge (\nexists q_s^B \in Q^B \cdot (q^B, s, q_s^B) \in \delta^B)$  alors
9:       (* on ne fait rien *)
10:      sinon
11:        si  $(\nexists q_s^A \in Q^A \cdot (q^A, s, q_s^A) \in \delta^A) \text{ xor } (\nexists q_s^B \in Q^B \cdot (q^B, s, q_s^B) \in \delta^B)$  alors
12:          retourner faux;
13:        fin si
14:        soient  $q_s^A = \delta^A(q^A, s)$  et  $q_s^B = \delta^B(q^B, s)$ 
           (*  $q_s^A$  et  $q_s^B$  sont les successeurs de  $q^A$  et  $q^B$  par  $s$  dans  $A$  et  $B$  *)
15:        si  $(q_s^A, q_s^B) \notin \text{Déjà\_visités} \cup \mathbb{A}_{\text{visiter}}$  alors
16:          si  $\left( q_s^A \in \left\{ q \mid (q, q') \in \text{Déjà\_visités} \cup \mathbb{A}_{\text{visiter}} \right\} \right) \vee$ 
              $\left( q_s^B \in \left\{ q' \mid (q, q') \in \text{Déjà\_visités} \cup \mathbb{A}_{\text{visiter}} \right\} \right)$  alors
               (* si l'un des deux états était déjà associé à un autre état *)
17:            retourner faux;
18:          sinon
19:             $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup \{(q_s^A, q_s^B)\}$ ;
               (* on ajoute le couple  $(q_s^A, q_s^B)$  dans l'ensemble des états à vérifier *)
20:          fin si
21:        fin si
22:      fin si
23:    fin pour
24:  fin tant que
25: retourner vrai;
```



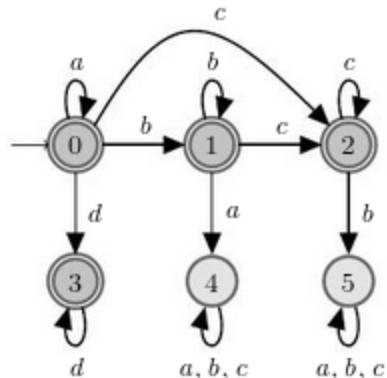
(a) AFED reconnaissant un langage préfixe-clos : le langage des mots contenant que des a .



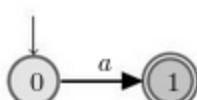
(b) AFED reconnaissant un langage préfixe-clos : le langage vide.



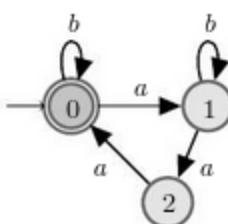
(c) AFED reconnaissant un langage préfixe-clos : le langage des mots contenant au plus deux occurrences de a .



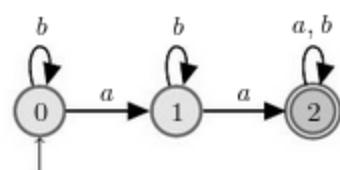
(d) AFED reconnaissant un langage préfixe-clos : le langage des mots qui soit contiennent que des occurrences du symbole d soit les mots formés par une succession d'occurrences des symboles a, b et c dans cet ordre et sans contrainte sur le nombre d'occurrences.



(e) AFED reconnaissant un langage non préfixe-clos : le langage contenant le mot a



(f) AFED reconnaissant un langage non préfixe-clos : le langage des mots contenant un nombre d'occurrences de a divisible par 3.



(g) AFED reconnaissant un langage non préfixe-clos : le langage des mots contenant au moins deux occurrences du symbole a .

Figure 5.4 – AFED pour l'exercice 55 (p. 103).

- L'AFED dans la figure 5.4c reconnaît le langage des mots ne contenant pas plus de deux occurrences du symbole a .
 - L'AFED dans la figure 5.4d reconnaît le langage des mots qui soit contiennent que des occurrences du symbole d soit les mots formés par une succession d'occurrences des symboles a, b et c dans cet ordre et sans contrainte sur le nombre d'occurrences.
 - Pour les langages non préfixe-clos, nous avons les automates suivants :
 - L'AFED dans la figure 5.4e reconnaît le langage contenant le mot a .
 - L'AFED dans la figure 5.4f reconnaît le langage des mots contenant un nombre d'occurrences du symbole a divisible par 3.
 - L'AFED dans la figure 5.4g reconnaît l'ensemble des mots contenant au moins deux occurrences du symbole a .
2. En observant les automates donnés dans la réponse précédente, nous observons les caractéristiques communes suivantes :
- dans les automates reconnaissant des langages préfixe-clos, il n'y a pas de transition depuis un état non accepteur vers un état accepteur;
 - inversement, dans les automates reconnaissant des langages non préfixe-clos, il y a au moins une transition depuis un état non accepteur vers un état accepteur.
- La présence ou l'absence de telles transitions conditionne respectivement la possibilité ou l'impossibilité pour un mot accepté d'avoir un préfixe non accepté.
3. Pour déterminer si un automate reconnaît un langage préfixe-clos, nous vérifions la présence ou l'absence d'une transition depuis un état non accepteur vers un état accepteur. Pour être pertinent, un tel état doit accessible. Nous en déduisons l'algorithme 26.
- L'algorithme précédent nécessite le calcul des états accessibles de l'automate, et donc un parcours complet de l'automate, dans tous les cas. Il est possible de combiner le parcours de l'automate et le test de la condition indiquée précédemment. Pour cela, nous nous inspirons de la version récursive de l'algorithme calculant l'ensemble des états accessibles. Nous modifions les deux parties comme suit. L'algorithme est paramétré maintenant par un booléen qui sert à se souvenir si lors du parcours, nous avons visité un état non accepteur lors d'un précédent appel récursif. Si cela est le cas, l'état que l'on visite ne doit pas être accepteur. Nous obtenons l'algorithme 27 et l'algorithme 28.
4. En exécutant ces deux algorithmes sur les automates reconnaissant des langages préfixe-clos, nous observons que les deux algorithmes retournent le booléen vrai.

Algorithme 26 *est_prefixe_clos()* pour déterminer si un automate reconnaît un langage préfixe-clos

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : vrai si $\mathcal{L}_{\text{auto}}(A)$ est préfixe-clos, faux sinon

- 1: Accessibles := *états_accessibleles*(A);
 - 2: **retourner** $\{q \in \text{Accessibles} \cap \overline{F} \mid \exists q' \in \text{Accessibles} \cap F. \exists s \in \Sigma. (q, s, q') \in \delta\} == \emptyset$;
-

Algorithme 27 *est_prefixe_clos_rec()* pour déterminer si l'automate reconnaît un langage préfixe-clos - version récursive - partie principale

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : vrai si le langage reconnu par l'automate est préfixe-clos, faux sinon

- 1: **ensemble d'états** Déjà_visités; (* variable globale aux deux algorithmes *)
 - 2: Déjà_visités := \emptyset ; (* initialement, rien n'est visité *)
 - 3: **retourner** *est_prefixe_clos_rec_interne*(q_{init} , faux);
-

Algorithme 28 *est_prefixe_clos_rec_interne()* pour déterminer si l'automate reconnaît un langage préfixe-clos - version récursive - partie récursive

Entrée : $q \in Q$ (* un état *)

Entrée : *vu* (* un booléen qui indique si on est passé par un état non accepteur *)

- 1: **si** *vu* $\wedge q \in F$ **alors**
 - 2: **retourner** faux;
 - 3: **fin si**
 - 4: Déjà_visités := Déjà_visités $\cup \{q\}$; (* l'état q n'est plus à visiter *)
 - 5: **pour** chaque état $q' \in \text{Succ}(q) \setminus \text{Déjà_visités}$ **faire**
 - 6: **si** ($q' \notin \text{Déjà_visités}$) $\wedge (\neg \text{est_prefixe_clos_rec_interne}(q', q \notin F))$ **alors**
 - 7: **retourner** faux;
 - 8: **fin si**
 - 9: **fin pour**
 - 10: **retourner** vrai;
-

Solution de l'exercice 56 (page 103)

1. En s'inspirant des résultats trouvés à l'exercice 55, nous observons qu'il faut supprimer du langage accepté par l'automate les mots qui ont un préfixe qui n'est pas dans le langage accepté par l'automate. Une manière simple de réaliser cela est d'assurer que lors de la lecture d'un symbole, l'automate passe d'un état accepteur à un état non accepteur, aucun des symboles que l'automate pourrait recevoir ne doit mener à un état accepteur. Ainsi, il est possible de simplement supprimer les transitions sortant des états accepteurs et menant vers un état non accepteur. Nous en déduisons l'algorithme 29.

Notons que l'application de cette algorithme a pour effet de supprimer des transitions de l'automate, modifiant l'accessibilité des états. Il est possible ainsi de « nettoyer » l'automate en utilisant l'algorithme de calcul des états accessibles pour supprimer les états de Q et les états de F qui seraient devenus non accessibles.

Algorithme 29 *ss_ens_prefixe_clos_rec()* pour réduire le langage d'un automate à son plus grand sous-ensemble préfixe-clos

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED *)

Sortie : L'AFED reconnaissant le plus grand sous-ensemble préfixe-clos du langage reconnu par A

```

1: si  $q_{\text{init}} \in F$  alors
2:   pour  $(q, s, q') \in \delta$  faire
3:     si  $q \in F \wedge q' \in F$  alors
4:        $\delta_{\text{pref}} := \delta_{\text{pref}} \cup \{(q, s, q')\};$ 
5:     fin si
6:   fin pour
7:   retourner  $(Q, \Sigma, q_{\text{init}}, \delta_{\text{pref}}, F);$ 
8: sinon
9:   retourner  $(\{q_{\text{init}}\}, \Sigma, q_{\text{init}}, \emptyset, \emptyset);$ 
10: fin si
```

Minimisation d'automates déterministes

6.1 Résumé intuitif du chapitre

Dans ce chapitre, nous nous intéressons d'une part à la notion d'**équivalence** entre automates déterministes et d'autre part à l'opération de **minimisation**. Pour simplifier la présentation des résultats, et sans perte de généralité, nous considérons des automates déterministes et complets, dont tous les états sont accessibles. Pour motiver l'importance de ces notions, nous considérons l'exemple d'automate donné dans la figure 6.1 (p. 120).

Nous pouvons nous poser les questions suivantes :

- Est-ce que dans cet automate certains états peuvent être "distingués", c'est-à-dire est-ce qu'il est possible de déterminer de manière certaine que certains états jouent un rôle différent dans l'acceptation de mot ?
- De manière duale, est-ce que certains états peuvent être caractérisés comme "équivalents", c'est-à-dire qu'à partir de ces états, les mêmes mots peuvent être acceptés ?

En examinant l'automate, on peut remarquer que les états 4 et 6 ont un "comportement" similaire. En effet, sur le symbole *a*, les transitions sortantes des états 4 et 6 mènent à l'état 3 sur le symbole *b* les transitions mènent à l'état 7. Ainsi, les mots qui seront acceptés à partir des états 4 et 6 seront les mêmes. Nous dirons que ce sont des **états équivalents**.

Nous pouvons faire la même remarque pour les états 2 et 8. Ces états sont équivalents. Or, d'une certaine manière, la notion d'équivalence se "propage". En effet, on peut observer que l'état 2 est atteint à partir de l'état 1 sur le symbole *a*, et l'état 8 est atteint à partir de l'état 5 sur le symbole *a*. Donc, les états 1 et 5 mènent à deux états équivalents sur le symbole *a* et au même état sur le symbole *b*. Les états 1 et 5 peuvent être caractérisés comme étant équivalents. Le fait de pouvoir caractériser des états comme équivalents ou non peut se faire pas-à-pas en regardant uniquement les états atteints sur une transition et en utilisant l'information sur les états que nous avons déjà caractérisés comme équivalents ou non. C'est ce principe qui est utilisé par les algorithmes étudiés dans ce chapitre.

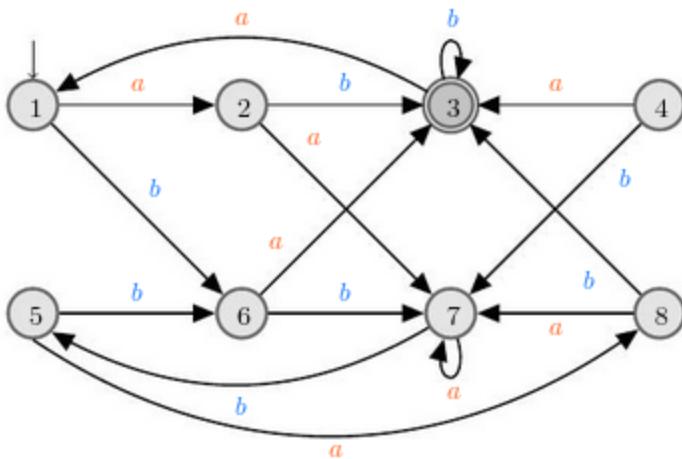


Figure 6.1 – Un automate pour lequel on cherche les états distinguables et les états équivalents

L'intérêt de caractériser deux états comme équivalents est qu'il est en fait utile d'en garder uniquement un des deux, pour pouvoir reconnaître le langage. Ceci permet d'avoir une représentation plus petite d'un automate en réduisant le nombre d'états et de transitions.

Plus généralement, nous nous intéressons à répondre aux questions suivantes :

- Peut-on définir formellement une relation d'équivalence (entre états) compatible avec la fonction de transition ?
- Peut-on dire si des automates sont équivalents (c'est-à-dire que les automates reconnaissent le même langage) ?
- Peut-on avoir une représentation "minimale" d'un automate ?

Comme nous l'avons vu de manière informelle, les notions d'équivalence et de distinguabilité sont liées à la notion d'acceptation de mot et langage (à partir d'un état). À partir de la notion d'acceptation, nous allons définir la notion d'équivalence entre états sous forme de relation entre états, et puis déterminer une manière effective de calculer cette relation. Sachant déterminer si deux états d'un automate sont équivalents, nous saurons déterminer si deux automates sont équivalents. Enfin, sachant partitionner les états d'un automate selon sa fonction de transition, nous pourrons calculer la représentation minimale de cet automate en supprimant les états redondants c'est-à-dire les états qui sont équivalents à d'autres états de l'automate. Plus précisément, nous appellerons cet automate qui reconnaît le même langage l'automate quotient ou l'**automate minimisé**.

6.2 Les notions essentielles

Dans ce chapitre, nous introduisons la notion de distinguabilité entre états et la relation de distinguabilité d'un automate (section 6.2.1), la relation d'équivalence entre états et la relation d'équivalence d'un automate (section 6.2.2). Nous utilisons ces notions pour définir la minimisation et l'automate minimisé (et minimal) (section 6.2.3). Enfin, nous définissons des

procédures pour déterminer si deux automates sont équivalents, c'est-à-dire s'ils reconnaissent le même langage (section 6.2.4).

Pour simplifier la présentation des résultats, nous considérons des AFED complets, dont tous les états sont accessibles. Cette hypothèse est sans perte de généralité car nous avons montré que la complétion préserve le langage d'un automate (propriété 1 et exercice 36) ainsi que l'opération d'émondage (exercice 51). Dans la suite, nous considérons un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

6.2.1 Relation de distinguabilité entre états

Relation de distinguabilité (générale).

Définition 97 (Relation de distinguabilité entre états) *La relation de distinguabilité entre états est une relation sur Q . Elle est notée $\not\equiv$ et définie par :*

$$\not\equiv = \{(q, q') \in Q \times Q \mid \exists u \in \Sigma^*. \delta^*(q, u) \in F \iff \delta^*(q', u) \in F\}.$$

Deux états q et q' sont dans la relation de distinguabilité entre états de l'automate s'il existe un mot $u \in \Sigma^*$ tel que les états atteints en lisant u à partir de q et q' sont tels qu'un seul d'entre eux (exactement) est accepteur. Lorsque pour deux états $q, q' \in Q$ et un mot $u \in \Sigma^*$, on a $\delta^*(q, u) \in F \iff \delta^*(q', u)$, on dit que q et q' sont distingués par u ; on dit également que q et q' sont distinguables et que le mot u est le « témoin » de leur distinguabilité.

Propriété 5 (Antiréflexivité et symétrie de la relation de distinguabilité) *La relation de distinguabilité $\not\equiv$ entre états de Q est antiréflexive et symétrique :*

- $\forall q \in Q. \neg(q \not\equiv q)$ (antiréflexivité);
- $\forall q, q' \in Q. q \not\equiv q' \implies q' \not\equiv q$ (symétrie).

Relation de distinguabilité à k pas. La définition de la relation de distinguabilité n'est pas utilisable directement pour déterminer tous les états distinguables d'un automate. En effet, pour que deux états soit distinguable il faut montrer l'existence d'un mot témoin sur l'alphabet de l'automate. Ce mot est de longueur non bornée. Ainsi, la recherche du témoin se fait dans l'ensemble infini des mots sur l'alphabet de l'automate. Pour cela, la relation de distinguabilité à k pas est introduite. La relation de distinguabilité à k pas permet de borner la longueur des mots (qui doivent être de longueur au plus k) et la recherche des mots se fait donc dans un ensemble fini. De plus, la relation de distinguabilité à k pas fournit une vision inductive de la distinguabilité amenant à un algorithme de calcul des états distinguables. Elle peut s'exprimer par récurrence : on peut exprimer la distinguabilité à $k + 1$ pas en fonction de la distinguabilité à k pas et ainsi réaliser le calcul de la relation de distinguabilité de manière itérative.

Définition 98 (Distinguabilité à k pas) *Pour chaque $k \in \mathbb{N}$, la relation de distinguabilité entre états à k pas est une relation sur Q . Elle est notée $\not\equiv_k$ et est définie par récurrence sur k comme suit :*

- $\not\equiv_0 = \{(q, q') \in Q \times Q \mid q \in F \iff q' \in F\};$
- $\not\equiv_{k+1} = \not\equiv_k \cup \{(q, q') \in Q \times Q \mid \exists s \in \Sigma. (\delta(q, s) \not\equiv_k \delta(q', s))\}.$

Remarque 26 La relation de distinguabilité entre états à k pas peut se définir de manière équivalente comme suit :

- $q \not\equiv_0 q'$ si et seulement si $q \in F \iff q' \in F$.
- Pour $k \in \mathbb{N}$, $q \not\equiv_{k+1} q'$ si et seulement si $(q \not\equiv_k q' \text{ ou } \exists s \in \Sigma. (\delta(q, s) \not\equiv_k \delta(q', s)))$.

Construction de $\not\equiv$ à partir de $\not\equiv_k$.

Lemme 1 (Longueur du mot témoin de la distinguabilité) Pour tout entier $k \in \mathbb{N}$, états $q, q' \in Q$, q et q' sont distinguables à k pas si et seulement s'ils sont distingués par un mot de longueur inférieure ou égale à k :

$$\forall k \in \mathbb{N}. \forall q, q' \in Q. (q \not\equiv_k q') \iff (\exists u \in \Sigma^*. |u| \leq k \wedge (\delta^*(q, u) \in F \iff \delta^*(q', u) \in F)).$$

Lemme 2 Pour tout $k \in \mathbb{N}$, les états distinguables à k pas sont distinguables à $k + 1$ pas :

$$\forall k \in \mathbb{N}. \not\equiv_k \subseteq \not\equiv_{k+1}.$$

Corollaire 4 L'union pour $k \in \mathbb{N}$ de tous les états distinguables à k pas est l'ensemble des états distinguables de l'automate :

$$\bigcup_{k \in \mathbb{N}} \not\equiv_k = \not\equiv.$$

Algorithme pour le calcul de $\not\equiv$.

Définition 99 (Algorithme pour le calcul des états distinguables) L'algorithme 30 permet de calculer la relation de distinguabilité entre états. L'algorithme prend en entrée un AFED complet $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ dont tous les états sont accessibles et produit en sortie sa relation de distinguabilité, c'est-à-dire l'ensemble des couples d'états distinguables. L'algorithme utilise trois ensembles de couples d'états D , D_{pre} et X . L'algorithme réalise une itération pour le calcul de $\not\equiv$ où la k -ème itération correspond au calcul de $\not\equiv_k$ et s'arrête à la découverte du point fixe, c'est-à-dire lorsque l'algorithme trouve que $\not\equiv_k = \not\equiv_{k-1}$. La variable D contient l'ensemble des couples d'états trouvés comme distinguables jusqu'à l'itération courante. La variable D_{pre} contient l'ensemble des couples d'états trouvés comme distinguables jusqu'à l'itération précédente (ou \emptyset à la première itération). X est une variable temporaire utilisée pour stocker les nouveaux états marqués comme distinguables à l'itération courante et ajoutés à l'ensemble des états distinguables.

Correction et complétude de l'algorithme de calcul de $\not\equiv$.

Théorème 7 (À propos de l'algorithme de distinction entre états) L'algorithme 30 (p. 123) de calcul des états distinguables est correct et complet :

- l'algorithme calcule uniquement les états distinguables selon $\not\equiv$ (correction),
 - l'algorithme calcule tous les états distinguables selon $\not\equiv$ (complétude),
- où $\not\equiv$ est la relation de distinguabilité calculée avec A .

Algorithme 30 Calcul des états distinguables

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED complet et accessible *)
Sortie : $\not\equiv \subseteq Q \times Q$ (* la relation de distinguabilité entre états de Q *)
ensemble de couples d'états D; (* D contient les états distinguables *)
ensemble de couples d'états D_{pre} ; (* D_{pre} est la valeur de D à l'itération précédente *)
ensemble de couples d'états X;
(* variable temporaire pour les nouveaux états distinguables d'une itération *)
 $D := F \times (Q \setminus F);$ (* D initialisé avec états terminaux/non terminaux *)
 $D_{\text{pre}} := \emptyset;$ (* initialisation de D_{pre} *)
tant que $D_{\text{pre}} \neq D$ **faire**
 $D_{\text{pre}} := D;$ (* mise à jour de D_{pre} /sauvegarde de D *)
 $X := \{(p, q) \in Q \mid \exists s \in \Sigma. (\delta(p, s), \delta(q, s)) \in D \vee (\delta(q, s), \delta(p, s)) \in D\};$
 (* calcul des nouveaux états distinguables *)
 $D := D \cup X;$ (* ajouter les nouveaux états distinguables à D *)
fin tant que
retourner D;

6.2.2 Relation d'équivalence entre états d'un AFED

Définition 100 (Relation d'équivalence entre états) La relation d'équivalence entre états est une relation sur Q . Elle est notée \equiv et définie par :

$$\equiv = \{(q, q') \in Q \times Q \mid \forall u \in \Sigma^*. (\delta^*(q, u) \in F \iff \delta^*(q', u) \in F)\}.$$

Deux états q et q' sont dans la relation d'équivalence entre états de l'automate si pour tout mot $u \in \Sigma^*$ les états atteints en lisant u à partir de q et q' sont soit tous deux accepteurs soit tous deux non accepteurs. Lorsque pour deux états $q, q' \in Q$ et pour un mot $u \in \Sigma^*$, on a $\delta^*(q, u) \in F \iff \delta^*(q', u)$, on dit que q et q' sont équivalents selon u ; lorsque cela est le cas pour tout mot $u \in \Sigma^*$, on dit que q et q' sont équivalents.

Propriété 6 (\equiv est une relation d'équivalence) La relation \equiv est effectivement une relation d'équivalence au sens de la définition 22 (p. 23) : elle est réflexive, symétrique et transitive.

Relation d'équivalence à k pas. Pour les mêmes raisons que nous avons introduit la relation de distinguabilité à k pas, nous introduisons la relation d'équivalence à k pas.

Définition 101 (Équivalence à k pas) Pour chaque $k \in \mathbb{N}$, la relation d'équivalence à k pas est une relation sur Q . Elle est notée \equiv_k et est définie par récurrence comme suit :

1. $q \equiv_0 q'$ si et seulement si $(q \in F \text{ et } q' \in F)$ ou $(q \notin F \text{ et } q' \notin F)$.
2. Pour $k \in \mathbb{N}$, $q \equiv_{k+1} q'$ si et seulement si $q \equiv_k q'$ et $\forall s \in \Sigma. (\delta(q, s) \equiv_k \delta(q', s))$.

Construction de \equiv à partir de \equiv_k .

Lemme 3 Pour tout $k \in \mathbb{N}$, états $q, q' \in Q$, q et q' sont équivalents à k pas si et seulement s'ils ne sont pas distingués pour tout mot de longueur inférieure ou égale à k :

$$\forall k \in \mathbb{N}. \forall q, q' \in Q.$$

$$(q \equiv_k q') \iff \forall u \in \Sigma^*. |u| \leq k \implies (\delta^*(q, u) \in F \iff \delta^*(q', u) \in F).$$

Corollaire 5 L'intersection pour $k \in \mathbb{N}$ de tous les états équivalents à k pas est l'ensemble des états équivalents de l'automate :

$$\bigcap_{k \in \mathbb{N}} \equiv_k = \equiv.$$

Définition 102 (Algorithm de calcul des classes d'équivalence) L'algorithme 31 permet de calculer la relation d'équivalence entre états. L'algorithme prend en entrée un AFED complet $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ dont tous les états sont accessibles et produit en sortie la relation d'équivalence \equiv pour A , c'est-à-dire l'ensemble des couples de la relation d'équivalence entre états de A . L'algorithme utilise trois ensembles de couples d'états R , R_{pre} et X . L'algorithme réalise une itération pour le calcul de $\not\equiv$ où la k -ème itération correspond au calcul de \equiv_k et s'arrête à la découverte du point fixe, c'est-à-dire lorsque l'algorithme trouve que $\equiv_k = \equiv_{k-1}$. La variable R contient l'ensemble des couples d'états qui sont toujours équivalents jusqu'à l'itération courante. La variable R_{pre} contient l'ensemble des couples d'états qui sont toujours équivalents jusqu'à l'itération précédente (ou \emptyset à la première itération). X est une variable temporaire utilisée pour stocker les nouveaux états marqués comme distinguables à l'itération courante et supprimés de l'ensemble des états équivalents.

Algorithme 31 Calcul des classes d'équivalence

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AFED complet et accessible *)
Sortie : $\equiv \subseteq Q \times Q$ (* la relation d'équivalence entre états de Q *)

- 1: **ensemble de couples d'états R ;** (* R contient les états équivalents *)
- 2: **ensemble de couples d'états R_{pre}** (* R_{pre} est la valeur de R à l'itération précédente *)
- 3: **ensemble de couples d'états X ;** (* temporaire pour les nouveaux états non équivalents lors d'une itération *)
- 4: $R := (F \times F) \cup ((Q \setminus F) \times (Q \setminus F))$; (* R initialisée à la partition entre états terminaux/non terminaux *);
- 5: $R_{\text{pre}} := \emptyset$; (* initialisation de R_{pre} *)
- 6: **tant que** $R_{\text{pre}} \neq R$ **faire**
- 7: $R_{\text{pre}} := R$; (* mise à jour de R_{pre} *)
- 8: $X := \{(p, q) \in R \mid \exists s \in \Sigma. (\delta(p, s), \delta(q, s)) \notin R\}$; (* calcul des états distinguables *);
- 9: $R := R \setminus X$; (* enlever les états distinguables de R *)
- 10: **fin tant que**
- 11: **retourner** R ;

Lien entre les relations de distinguabilité et d'équivalence. Nous remarquons la dualité entre les définitions et les propriétés de la distinguabilité et l'équivalence entre états.

Théorème 8 (Dualité entre équivalence et distinguabilité)

$$\forall p, q \in Q. p \not\equiv q \iff \neg(p \equiv q)$$

Le théorème indique que deux états sont distinguables si et seulement s'ils ne sont pas équivalents.

6.2.3 Minimisation

Dans la suite, nous notons :

- $[q]_{\equiv} = \{q' \in Q \mid q \equiv q'\}$: la classe d'équivalence de l'état q ;
- $[Q]_{\equiv} = \{[q]_{\equiv} \mid q \in Q\}$: l'ensemble des classes d'équivalence de l'ensemble d'états Q , aussi appelé ensemble quotient de Q par \equiv .

Définition 103 (Automate minimisé/quotient) *Le minimisé de A ou encore automate quotient de A , noté A/\equiv , est l'automate*

$$\left(Q'^{\equiv}, \Sigma, q'^{\equiv}_{\text{init}}, \delta'^{\equiv}, F'^{\equiv} \right)$$

tel que :

- $Q'^{\equiv} = [Q]_{\equiv}$;
- $q'^{\equiv}_{\text{init}} = [q_{\text{init}}]_{\equiv}$;
- δ'^{\equiv} est la fonction de transition telle que :

$$\begin{array}{ccc} \delta'^{\equiv} & : & Q'^{\equiv} \times \Sigma \rightarrow Q'^{\equiv} \\ \delta'^{\equiv}([q], s) & = & [\delta(q, s)] \end{array}$$

- $F'^{\equiv} = \{[q] \in Q'^{\equiv} \mid q \in F\}$

Remarque 27 Le minimisé est aussi appelé automate quotient.

Dans la suite, nous considérons $A'^{\equiv} = \left(Q'^{\equiv}, \Sigma, q'^{\equiv}_{\text{init}}, \delta'^{\equiv}, F'^{\equiv} \right)$, le minimisé de A .

Lemme 4

$$\delta'^{\equiv}(X, s) = Y \iff \exists q \in X. \exists q' \in Y. \delta(q, s) = q'$$

Théorème 9 (Correction du minimisé) *Un automate et son minimisé reconnaissent le même langage.*

$$\mathcal{L}_{\text{auto}}(A'^{\equiv}) = \mathcal{L}_{\text{auto}}(A).$$

Définition 104 (Minimalité) *Un automate complet est dit minimal (pour le langage qu'il reconnaît) s'il n'existe pas d'automate complet qui reconnaît le même langage avec strictement moins d'états.*

Théorème 10 (Minimalité du minimisé) A'^{\equiv} est minimal (pour $\mathcal{L}_{\text{auto}}(A)$).

6.2.4 Tester l'équivalence entre deux AFED

Nous définissons des procédures permettant de déterminer si deux automates sont équivalents, c'est-à-dire s'ils reconnaissent le même langage. Nous considérons deux AFED $A = (Q^A, \Sigma, q_{\text{init}}^A, \delta^A, F^A)$, $B = (Q^B, \Sigma, q_{\text{init}}^B, \delta^B, F^B)$ sur le même alphabet Σ .

En utilisant la distinguabilité entre états

Sans perte de généralité, supposons que les ensembles d'états des deux automates que nous comparons sont disjoints (c'est-à-dire que $Q^A \cap Q^B = \emptyset$).

Propriété 7 (Équivalence en utilisant la distinguabilité) *Les AFED A et B sont équivalents si et seulement si dans l'automate $(Q^A \cup Q^B, \Sigma, q_{\text{init}}^A, \delta^A \cup \delta^B, F^A \cup F^B)$, les états q_{init}^A et q_{init}^B ne sont pas distinguables.*

En utilisant la minimisation

Définition 105 (Renommage des états d'un automate) *Une fonction de renommage des états de A vers les états de B est une fonction bijective $f : Q^A \rightarrow Q^B$ telle que :*

- $Q^B = \{f(q) \mid q \in Q^A\}$,
- $q_{\text{init}}^B = f(q_{\text{init}}^A)$,
- $\delta^B = \{(f(q), s, f(q')) \mid (q, s, q') \in \delta^A\}$,
- $F^B = \{f(q) \mid q \in F^A\}$.

Remarque 28 *L'existence une fonction de renommage des états de A vers les états de B implique l'existence d'une fonction de renommage des états de B vers les états A.*

Propriété 8 (Équivalence en utilisant un renommage des états) *Les AFED A et B sont équivalents s'il existe une fonction de renommage des états de A vers B. De plus, si les AFED A et B sont minimaux, les AFED A et B sont équivalents si et seulement s'il existe une fonction de renommage des états de A vers B. L'algorithme 25 (p. 114) trouvé en solution de l'exercice 53 (p. 102) teste l'existence d'une fonction de renommage entre deux AFED.*

En utilisant le produit pour l'intersection.

Propriété 9 (Équivalence en utilisant le produit pour l'intersection) *Les AFED A et B sont équivalents si et seulement si $\mathcal{L}_{\text{auto}}(A \times B^c) = \mathcal{L}_{\text{auto}}(A^c \times B) = \emptyset$, où A^c et B^c sont les automates complémentaires des automates A et B selon la définition 79 (p. 72) et \times est l'opération de produit entre automates selon la définition 80 (p. 73).*

6.3 Exercices

6.3.1 Équivalence et distinguabilité entre états

Exercice 57 (♠) — États distinguables et états équivalents

Nous considérons l'automate dans la figure 6.1 (p. 120) sur l'alphabet $\Sigma = \{a, b\}$.

1. Indiquer les états équivalents de cet automate.
2. Indiquer les états distinguables de cet automate.

6.3.2 Minimisation

Exercice 58 (♠♠♠) — Trouver l'automate minimal reconnaissant un langage

Nous considérons l'alphabet $\Sigma = \{a, b\}$.

1. Donner un AFED minimal qui reconnaît le langage des mots sur l'alphabet Σ qui ne contiennent pas le facteur $a \cdot b \cdot a$.

Exercice 59 (♠♠) — Minimiser des automates

Nous considérons les automates dans la figure 6.2 sur l'alphabet $\Sigma = \{a, b\}$.

1. Minimiser l'AFED dans la figure 6.2a.
2. Minimiser l'AFED dans la figure 6.2b.

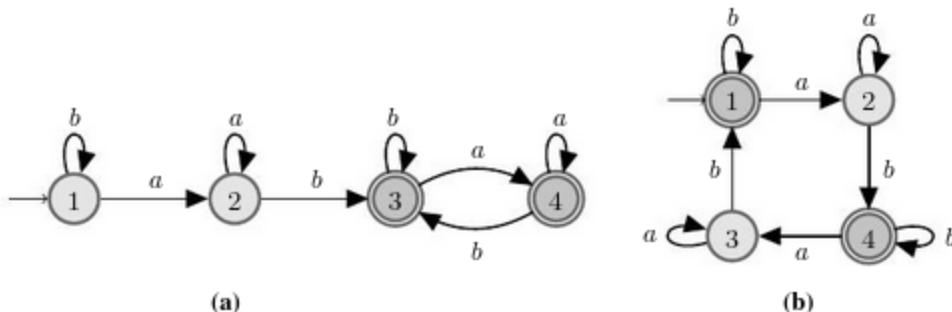


Figure 6.2 – Des AFED en représentation graphique à minimiser.

Exercice 60 (♠♠) — Déterminer si un automate est minimal ou non

Nous considérons les deux automates représentés sous forme graphique dans la figure 6.3. Nous souhaitons déterminer si ces automates sont minimaux. Pour chacun des automates (figure 6.3a et figure 6.3b), faire les questions suivantes.

1. Utiliser l'algorithme calculant les états distinguables pour déterminer la minimalité. Lister les classes d'équivalences.
2. Utiliser l'algorithme calculant les états équivalents pour déterminer la minimalité. Montrer clairement les étapes de calcul. Lister les classes d'équivalences.
3. Vérifier que vous obtenez des résultats compatibles et donc les mêmes conclusions avec les deux méthodes.
4. Si ces automates ne sont pas minimaux, représenter leur automate minimisé sous formes tabulaire et graphique.

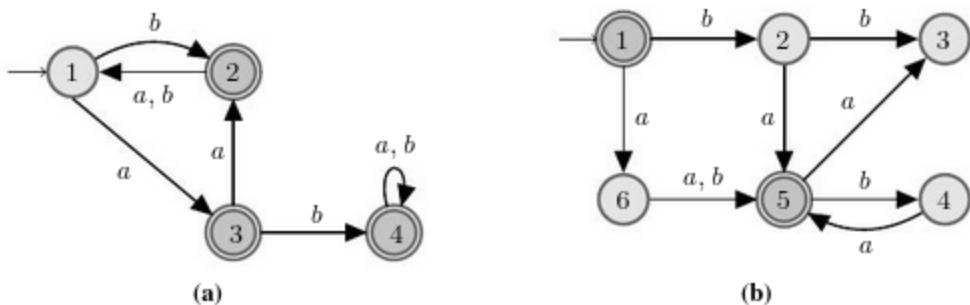


Figure 6.3 – Des AFED en représentation graphique à minimiser.

Exercice 61 (♣♣) — Déterminer si un automate est minimal ou non

Nous considérons les automates représentés sous forme tabulaire dans le tableau 6.1. Nous souhaitons déterminer si ces automates sont minimaux. Pour chacun des automates (tableau 6.1a, tableau 6.1b et tableau 6.1c) :

1. Utiliser l'algorithme calculant les états distinguables pour déterminer la minimalité.
2. Utiliser l'algorithme calculant les états équivalents pour déterminer la minimalité.
3. Vérifier que vous obtenez des résultats compatibles et donc les mêmes conclusions avec les deux méthodes.
4. Si ces automates ne sont pas minimaux, représenter leur automate minimisé sous formes tabulaire et graphique.

Tableau 6.1 – Des AFED en représentation tabulaire à minimiser. Les états sont en lignes, les symboles en colonnes. La flèche indique l'état initial, l'étoile indique un état terminal.

	a	b
→ 0	0	1
1	2	3
2	2	3
*3	2	4
4	0	1

(a)

	a	b
→ 1	3	8
*2	3	1
3	8	2
*4	5	6
5	6	2
6	7	8
7	6	4
8	5	8

(b)

	a	b
→ *0	1	3
1	2	3
*2	5	2
3	4	1
*4	5	4
5	5	5

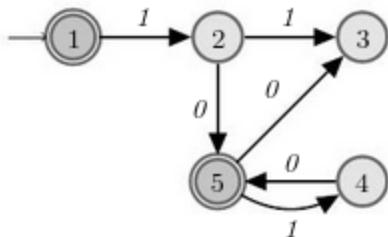
(c)

6.3.3 Équivalence et distinguabilité entre AFED

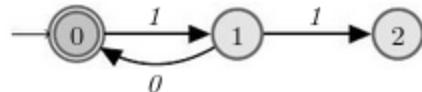
Exercice 62 (♠♦) — Déterminer l'équivalence ou la non-équivalence entre automates

Considérons les deux automates dans les figures 6.4a et 6.4b. Montrer que ces deux automates sont équivalents :

1. en utilisant la procédure basée sur la distinguabilité entre états ;
2. en utilisant la procédure basée sur la minimisation ;
3. en utilisant la procédure basée sur le produit d'automates.



(a)



(b)

Figure 6.4 – Des AFED dont on veut déterminer l'équivalence.

Exercice 63 (♠♦) — Montrer l'inclusion entre deux langages en utilisant leur automates

Considérons l'alphabet $\Sigma = \{a, b\}$.

1. Soit L_1 le langage des mots qui ne contiennent pas $abaa$ et qui contiennent un nombre pair de a . Donner un AFED complet et minimal qui reconnaît L_1 .
2. Soit L_2 le langage des mots qui ne contiennent pas aba et qui contiennent un nombre de a multiple de 4. Donner un AFED et minimal qui reconnaît L_2 .
3. Montrer que $L_2 \subseteq L_1$ en utilisant les résultats de l'exercice 52 (p. 102).

6.4 Indications pour résoudre les exercices

Indications pour l'exercice 62 (p. 129)

3. Réutiliser le résultat de l'exercice 52.

6.5 Solutions des exercices

6.5.1 Équivalence et distinguabilité entre états

Solution de l'exercice 57 (page 126)

- Les états équivalents sont comme suit : $4 \equiv 6$, $2 \equiv 8$, mais aussi $1 \equiv 5$ et $x \equiv x$ pour tout état x .
- Toutes les autres paires d'états représentent des états distinguables, c'est-à-dire $1 \not\equiv 2$, $1 \not\equiv 3$, $1 \not\equiv 4$, $1 \not\equiv 6$, $2 \not\equiv 3$, ...

6.5.2 Minimisation

Solution de l'exercice 58 (page 127)

- Définissons d'abord un automate qui reconnaît les mots qui ne contiennent pas aba (figure 6.5a).

Cet automate est complet. Maintenant, pour déterminer si cet automate est minimal, nous appliquons l'algorithme de minimisation. Le résultat final est représenté dans la figure 6.5b. Chaque colonne représente une étape de l'exécution de l'algorithme. À chaque étape, les états de l'automate sont listés verticalement. Les traits horizontaux représentent les séparations entre classes d'équivalence. Par exemple dans la deuxième colonne de la figure 6.5b correspondant au calcul de \equiv_1 , nous avons trois classes d'équivalence, une regroupant les états 0 et 1, une avec l'état 2 seul et une avec l'état 3 seul.

Dans la dernière colonne, chaque classe contient un état, l'automate est donc minimal. Nous détaillons étape par étape l'obtention de ce résultat.

Calcul de \equiv_0 À l'étape 0 (initialisation de l'algorithme), nous calculons \equiv_0 , c'est-à-dire les paires d'états qui sont équivalents avec les mots de longueur 0. Ainsi, nous partitionons les états entre états accepteurs et non accepteurs. Nous obtenons le résultat intermédiaire représenté dans la figure 6.5c.

Pour les étapes suivantes, à chaque étape i (avec $i > 0$), nous construisons \equiv_i , c'est-à-dire les paires d'états qui sont équivalents avec les mots de longueur i . Suivant

	a	b
$\rightarrow *0$	1	0
*1	1	2
*2	3	0
3	3	3

(a) AFED qui reconnaît les mots qui ne contiennent pas aba .

$\equiv 0$	$\equiv 1$	$\equiv 2$
0	0	0
1	1	1
2	2	2
3	3	3

(b) Minimisation de l'automate dans la figure 6.5a.

$\equiv 0$
0
1
2
3

(c) \equiv_0 .

$\equiv 0$	$\equiv 1$
0	0
1	1
2	2
3	3

(d) \equiv_1 .

Figure 6.5 – Résultats pour l'exercice 58 (p. 127).

l'algorithme, deux états peuvent être distingués par une séquence de longueur i si l'une des deux conditions suivantes est remplie :

- avec un symbole donné, selon la fonction de transition, les états successeurs sont dans des classes différentes selon \equiv_{i-1} (c'est-à-dire, à l'étape précédente) ;
- ils sont déjà distingués selon \equiv_{i-1} .

Calcul de \equiv_1 Intéressons-nous d'abord à la première classe d'équivalence représentée dans le tableau, c'est-à-dire $\{1, 2, 3\}$. Pour chaque état q de la classe d'équivalence, nous allons voir si il peut être distingué d'un autre état $q' \neq q$ (en utilisant la fonction de transition de l'automate).

- Considérons d'abord l'état 0. Nous considérons chaque symbole de l'alphabet.
 - En lisant a depuis l'état 0, l'automate se déplace dans l'état 1 qui est dans la classe d'équivalence $\{0, 1, 2\}$. Depuis l'état 1, en lisant a , l'automate va dans l'état 1, qui est dans la classe d'équivalence $\{0, 1, 2\}$: ainsi les états 0 et 1 ne peuvent pas être distingués par a . Depuis l'état 2, en lisant a , l'automate se déplace vers l'état 3, qui est dans la classe d'équivalence $\{3\}$: ainsi 0 et 2 peuvent être distingués par a . Ces états seront placés dans deux classes d'équivalence différentes.
 - En lisant b depuis l'état 0, l'automate va dans l'état 0 qui est dans la classe d'équivalence $\{1, 2, 3\}$. Depuis l'état 1, en lisant b , nous allons dans l'état 2, qui est dans la classe d'équivalence $\{1, 2, 3\}$: ainsi les états 0 et 1 ne peuvent pas être distingués avec b . Depuis l'état 2, en lisant b , l'automate va dans l'état 0, qui est dans la classe d'équivalence $\{1, 2, 3\}$: ainsi les états 0 et 2 ne peuvent pas être distingués par b . Notons que nous n'avons pas vraiment besoin de considérer si nous pouvons ou pas distinguer les états 0 et 2 par b car nous savons déjà qu'ils sont distinguables par a .
- Considérons l'état 1. Nous savons déjà qu'il ne peut pas être distingué de l'état 0. De plus, nous savons que les états 0 et 2 peuvent être distingués, donc nous avons déjà le résultat final. Dans d'autres situations (par exemple, si nous avions plus d'états dans l'automate) nous devrions vérifier si l'état 1 est distinguable ou équivalent à d'autres états.

Finalement, pour la partition \equiv_1 , nous avons les classes d'équivalence suivantes : $\{0, 1\}, \{2\}, \{3\}$. Nous représentons le résultat de cette analyse dans la figure 6.5d.

Calcul de \equiv_2 Nous avons une seule classe qui n'est pas un singleton, la classe $\{0, 1\}$. Déterminons si les états 0 et 1 peuvent être distingués.

- En lisant a depuis l'état 0, l'automate va dans l'état 1, qui est dans la classe d'équivalence $\{0, 1\}$ de l'étape précédente. En lisant a depuis l'état 1, nous allons dans l'état 1. Ainsi, les états 0 et 1 ne peuvent pas être distingués avec a .
- En lisant b depuis l'état 0, l'automate va dans l'état 0, qui est dans la classe d'équivalence $\{0, 1\}$. En lisant b depuis l'état 1, l'automate va dans l'état 2,

qui est dans la classe d'équivalence $\{2\}$. Ainsi, les états 0 et 1 peuvent être distingués avec b .

Nous obtenons le résultat final, qui est le tableau dans la figure 6.5b.

Rappelons que l'algorithme de minimisation termine soit lorsque chaque classe est un singleton (et tous les états sont distingués) soit si pour une étape i lors du calcul de \equiv_i , la partition obtenue est la même qu'à l'étape précédente, c'est-à-dire à l'étape où nous avons calculé \equiv_{i-1} .

Solution de l'exercice 59 (page 127)

1. Considérons l'automate représenté dans la figure 6.2a (p. 127). Notons que l'automate est complet. Nous appliquons l'algorithme de minimisation. Le résultat de l'exécution de l'algorithme de minimisation et de calcul des partitions est représenté dans la figure 6.6a.

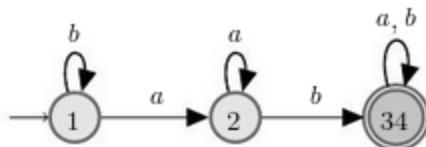
L'algorithme termine lors du calcul de \equiv_2 car la partition associée à \equiv_2 est égale à celle associée à \equiv_1 ; c'est-à-dire que plus aucun état n'a été distingué. L'automate n'est pas minimal : les états 3 et 4 peuvent être fusionnés. L'automate minimal est représenté dans la figure 6.6b où l'état 34 représente la fusion des états 3 et 4.

2. Considérons l'automate représenté dans la figure 6.2b. Notons que l'automate est complet. Nous appliquons l'algorithme de minimisation. Le résultat de l'exécution de l'algorithme de minimisation et de calcul des partitions est représenté dans la figure 6.6c.

L'algorithme termine lors du calcul de \equiv_1 car la partition associée à \equiv_1 est égale à celle associée à \equiv_0 ; c'est-à-dire que plus aucun état n'a été distingué. L'automate n'est pas minimal : les états 1 et 4 d'une part et 2 et 3 d'autre part peuvent être fusionnés. L'automate minimal est représenté dans la figure 6.6d où l'état 14 représente la fusion des états 1 et 4 et l'état 23 représente la fusion des états 2 et 3.

\equiv_0	\equiv_1	\equiv_2
1	1	1
2	2	2
3	3	3
4	4	4

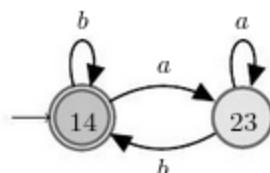
(a) Minimisation de l'AFED dans la figure 6.2a.



(b) Minimisé de l'AFED dans la figure 6.2a.

\equiv_0	\equiv_1
2	2
3	3
1	1
4	4

(c) Minimisation de l'AFED dans la figure 6.2b.



(d) Minimisé de l'AFED dans la figure 6.2b.

Figure 6.6 – Résultats pour l'exercice 59 (p. 127).

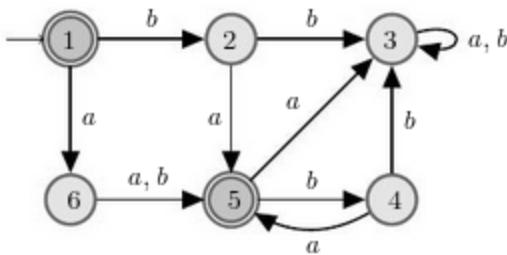
Solution de l'exercice 60 (page 127)

Pour appliquer les algorithmes dans les questions suivantes, nous devons utiliser des automates complets. Le premier automate est déjà complet. Le second automate n'est pas complet, nous le complétons avant d'appliquer l'algorithme. Pour réduire le nombre d'état, nous choisissons d'utiliser l'état 3, comme état puits dans cet automate. Compléter l'automate revient donc à ajouter une transition depuis l'état 3 vers lui même sur les symboles a et b et une transition depuis l'état 4 vers l'état 3 sur le symbole b . La version complétée du deuxième automate est représentée dans la figure 6.7a.

1. Nous appliquons l'algorithme de calcul des états distinguables à chaque automate. Nous représentons les résultats dans la figure 6.7b pour l'AFED dans la figure 6.3a et dans la figure 6.7c pour l'AFED dans la figure 6.3b. Pour l'AFED dans la figure 6.3a, nous indiquons ci-dessous les résultats des trois itérations qui donnent $\not\equiv_0$, $\not\equiv_1$ et $\not\equiv_2$ (respectivement figure 6.7d, figure 6.7e et figure 6.7f). Pour l'AFED dans la figure 6.3a, les états sont distinguables deux à deux, ces deux automates sont minimaux. Les classes d'équivalence sont les ensembles singletons formés par les états de l'automate : les ensembles de classes d'équivalence sont $\{\{x\} \mid x \in \{1, 2, 3, 4\}\}$ pour le premier automate et $\{\{x\} \mid x \in \{1, 2, 3, 4, 5, 6\}\}$ pour le second automate. Pour l'AFED dans la figure 6.3b, seuls les états 2 et 4 ne sont pas distinguables, les autres états étant distinguables deux à deux.
2. Nous appliquons l'algorithme de minimisation sur les deux automates. Nous représentons les résultats dans la figure 6.7g pour l'AFED dans la figure 6.3a et dans la figure 6.7h pour l'AFED dans la figure 6.3b.
Pour l'AFED dans la figure 6.3a, l'algorithme termine après le calcul de \equiv_2 où chaque classe d'équivalence est de cardinalité 1 (singleton). Le premier automate est donc minimal. Pour l'AFED dans la figure 6.3b, l'algorithme termine après le calcul de \equiv_3 qui est égale à \equiv_2 . L'automate n'est donc pas minimal, les états 2 et 4 peuvent être fusionnés.
3. Nous trouvons bien des résultats compatibles avec les deux algorithmes. Pour le premier automate, l'algorithme de calcul des états distinguables trouve que tous les états sont distinguables deux à deux, alors que l'algorithme de calcul des états équivalents place chaque état dans une classe d'équivalence de cardinalité 1. Pour le second automate, l'algorithme de calcul des états distinguables distingue tous les états deux à deux sauf les états 2 et 4 entre eux, alors que l'algorithme de calcul des états équivalents place les états 2 et 4 dans la même classe d'équivalence et chacun des autres états dans des classes d'équivalence de cardinalité 1.
4. La version minimisée de l'AFED dans la figure 6.3b est représentée sous forme tabulaire et graphique dans la figure 6.7i et figure 6.7j, respectivement. La version minimisée est obtenue en fusionnant les états dans les mêmes classes d'équivalence et en adaptant la fonction de transition des automates. Pour la version tabulaire, nous représentons les états en ligne et les symboles en colonnes, les états accepteurs sont indiqués par une étoile et l'état initial par une flèche.

Solution de l'exercice 61 (page 128)

Les automates considérés sont complets, nous pouvons donc appliquer les algorithmes directement.



(a) Version complétée de l'automate dans la figure 6.3b.

2	X			
3	X	X		
4	X	X	X	

1 2 3

(b) Résultat de l'algorithme de calcul des états distinguables sur l'AFED dans la figure 6.3a.

2	X			
3	X			
4	X			

1 2 3

(d) Étape 0 du calcul des états distinguables sur l'AFED dans la figure 6.3a ($\not\equiv_0$).

$\equiv 0$	$\equiv 1$	$\equiv 2$	$\equiv 3$
1	1	1	
2	2	2	
3	3	3	
4	4	4	

(g) Résultat de l'algorithme de minimisation sur l'AFED dans la figure 6.3a.

	$\downarrow 1*$	24	3	$5*$	6
a	6	5	3	3	5
b	24	3	3	24	5

(i) Minimisé de l'AFED dans la figure 6.3b - représentation tabulaire.

2	X			
3	X	X		
4	X		X	
5	X	X	X	X
6	X	X	X	X

1 2 3 4 5

(c) Résultat de l'algorithme de calcul des états distinguables sur l'AFED dans la figure 6.3b.

2	X			
3	X	X		
4	X	X		

1 2 3

2	X			
3	X	X		
4	X	X	X	

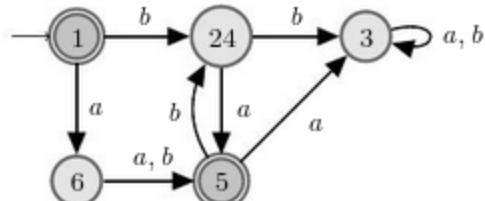
1 2 3

(e) Étape 1 du calcul des états distinguables sur l'AFED dans la figure 6.3a ($\not\equiv_1$).

$\equiv 0$	$\equiv 1$	$\equiv 2$	$\equiv 3$
1	1	1	1
5	5	5	5
2	2	2	2
3	4	4	4
4	3	3	3
6	6	6	6

(f) Étape 2 du calcul des états distinguables sur l'AFED dans la figure 6.3a ($\not\equiv_2$).

(h) Résultat de l'algorithme de minimisation sur l'AFED dans la figure 6.3b.



(j) Minimisé de l'AFED dans la figure 6.3b - représentation graphique.

Figure 6.7 – Résultats pour l'exercice 60 (p. 127). Dans les représentations du résultat de l'algorithme de calcul des états distinguables (figure 6.7b à figure 6.7f), nous représentons la relation de distinguabilité par un demi tableau (car la relation est symétrique), sans la diagonale (car la relation est antiréflexive), et tel que chaque croix indique les deux états de la ligne et colonnes correspondantes en relation (c'est-à-dire distinguables).

1. Nous appliquons l'algorithme de calcul des états distinguables à chaque automate. Nous représentons les résultats de l'algorithme de calcul des états distinguables dans la figure 6.8a (resp. figure 6.8b, figure 6.8c) pour l'AFED dans le tableau 6.1a (resp. tableau 6.1b, tableau 6.1c).
 - Dans la figure 6.8a, les états 0 et 4 d'une part et 1 et 2 d'autre part sont équivalents deux à deux. Les autres états sont distinguables deux à deux.
 - Dans la figure 6.8b, les états 1, 6 et 8 d'une part et 3, 5 et 7 d'autre part sont équivalents deux à deux. Les autres états sont distinguables deux à deux.
 - Dans la figure 6.8c, les états 2 et 4 d'une part et 1 et 3 d'autre part sont équivalents deux à deux. Les autres états sont distinguables deux à deux.
2. Nous appliquons l'algorithme de minimisation sur les trois automates.
 - Pour l'AFED du tableau 6.1a, voir figure 6.8d, l'algorithme termine après le calcul de \equiv_2 qui est égale à \equiv_1 . Il y a trois classes d'équivalence : $\{0, 4\}$, $\{1, 2\}$ et $\{3\}$. Cet AFED n'est donc pas minimal.
 - Pour l'AFED du tableau 6.1b, voir figure 6.8e, il y a trois classes d'équivalence : $\{1, 6, 8\}$, $\{3, 5, 7\}$ et $\{2, 4\}$.
 - Pour l'AFED du tableau 6.1c, voir figure 6.8f, il y a quatre classes d'équivalence : $\{1, 3\}$, $\{5\}$, $\{0\}$ et $\{2, 4\}$.
3. De manière similaire à l'observation faite dans la solution de l'exercice 60, nous observons que les états indiqués comme non distinguables par l'algorithme de distinguabilité sont indiqués comme étant dans la même classe d'équivalence par l'algorithme de calcul des classes d'équivalence entre états.
4. Nous obtenons les automates minimaux en fusionnant les états dans les mêmes classes d'équivalence et en adaptant la fonction de transition des automates. Les versions minimisées des trois automates sont représentées sous forme tabulaire (de la figure 6.8g à la figure 6.8i) et graphique (de la figure 6.8j à la figure 6.8l).

6.5.3 Équivalence et distinguabilité

Solution de l'exercice 62 (page 129)

Les algorithmes utilisés dans cet exercice requièrent de compléter les deux automates. Pour limiter le nombre d'états, nous choisissons, sans changer l'effet de la transformation de complétion, de choisir les états 3 et 8 comme états puits pour les deux automates de départ car il n'y a pas de transition dans les automates ayant ces états comme sources.

1. Nous appliquons la procédure basée sur la distinguabilité des états initiaux en construisant l'automate union, représenté dans la figure 6.9a. L'automate union, après complétion des deux automates de départ, est représenté ci-dessous. Par ailleurs, nous choisissons arbitrairement que l'état initial de l'automate union est l'état initial du premier automate.

Dans cet automate, nous devons tester si les deux états correspondant aux états initiaux des deux premiers automates (états 1 et 6) sont distinguables. Nous appliquons donc l'algorithme de distinguabilité dont les différentes itérations durant son exécution

1	X
2	X
3	X X X
4	X X X X

0 1 2 3

(a) Application de l'algorithme de calcul des états distinguables sur l'AFED du tableau 6.1a.

2	X
3	X X
4	X X X
5	X X X X
6	X X X X X
7	X X X X X X
8	X X X X X X X

1 2 3 4 5 6 7

(b) Application de l'algorithme de calcul des états distinguables sur l'AFED du tableau 6.1b.

1	X
2	X X
3	X X X
4	X X X X
5	X X X X X

0 1 2 3 4

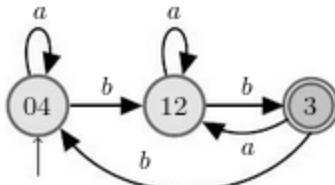
(c) Application de l'algorithme de calcul des états distinguables sur l'AFED du tableau 6.1c.

$\equiv 0$	$\equiv 1$	$\equiv 2$
0	0	0
1	4	4
2	2	2
4	1	1
3	3	3

(d) Application de l'algorithme de minimisation sur l'AFED du tableau 6.1a.

	$\downarrow 04$	12	$3*$
a	04	12	12
b	12	3	04

(g) Minimisé de l'AFED du tableau 6.1a - représentation tabulaire.



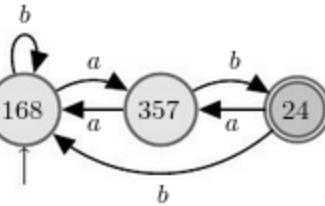
(j) Minimisé de l'AFED du tableau 6.1a - représentation graphique.

$\equiv 0$	$\equiv 1$	$\equiv 2$
1	1	1
3	6	6
5	8	8
6	3	3
7	5	5
8	7	7
2	2	2
4	4	4

(e) Application de l'algorithme de minimisation sur l'AFED du tableau 6.1b.

	$\downarrow 168$	357	$24*$
a	357	168	357
b	168	24	168

(h) Minimisé de l'AFED du tableau 6.1b - représentation tabulaire.



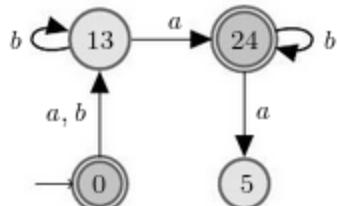
(k) Minimisé de l'AFED du tableau 6.1b - représentation graphique.

$\equiv 0$	$\equiv 1$	$\equiv 2$
1	1	1
3	3	3
5	5	5
0	0	0
2	2	2
4	4	4

(f) Application de l'algorithme de minimisation sur l'AFED du tableau 6.1c.

	$\downarrow 0*$	5	13	$24*$
a	13	5	24	5
b	13	5	13	24

(i) Minimisé de l'AFED du tableau 6.1c - représentation tabulaire.



(l) Minimisé de l'AFED du tableau 6.1c - représentation graphique.

Figure 6.8 – Résultats pour l'exercice 61 (p. 128). Pour la représentation tabulaire des automates, nous représentons les états en ligne et les symboles en colonnes, les états accepteurs sont indiqués par une étoile et l'état initial par une flèche.

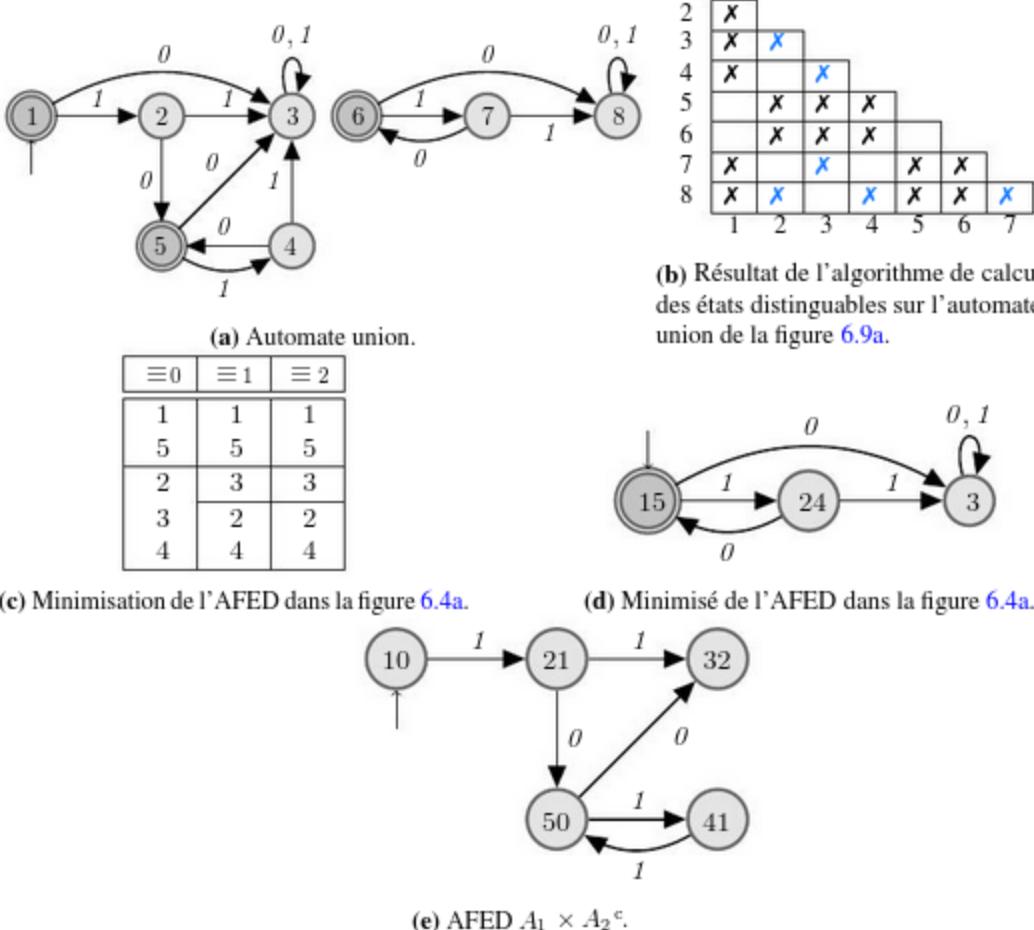


Figure 6.9 – Résultats pour l'exercice 62 (p. 129).

sont représentées dans la figure 6.9b. Les paires d'états marquées par une croix noire (X) sont celles marquées lors de l'étape d'initialisation ; elles correspondent à des paires constituées d'un état accepteur et d'un état non accepteur. Les paires d'états marquées par une croix bleue ($\textcolor{blue}{X}$) sont celles marquées lors de la première itération de l'algorithme. Lors de la seconde itération de l'algorithme, on ne peut plus distinguer d'états. L'algorithme termine. Nous observons que les états correspondant aux états initiaux utilisés dans la construction de cet automate (union) ne sont pas distinguables et nous concluons que les deux automates reconnaissent le même langage.

- Minimisons le premier automate en appliquant l'algorithme de minimisation. L'exécution de l'algorithme de minimisation est représentée dans la figure 6.9c et l'automate minimisé dans la figure 6.9d.

Nous remarquons immédiatement que la version minimisée du premier automate est le même (syntaxiquement) que le deuxième automate. Ainsi, la fonction de renom-

image des états permettant de démontrer que les deux automates sont syntaxiquement équivalents est la fonction identité.

3. Appelons A_1 et A_2 les deux automates donnés dans l'énoncé. Pour montrer que les deux automates sont équivalents, nous devons montrer que $\mathcal{L}_{\text{auto}}(A_1) = \mathcal{L}_{\text{auto}}(A_2)$, ce qui est équivalent à $\mathcal{L}_{\text{auto}}(A_1) \subseteq \mathcal{L}_{\text{auto}}(A_2)$ et $\mathcal{L}_{\text{auto}}(A_2) \subseteq \mathcal{L}_{\text{auto}}(A_1)$. Nous réutilisons pour cela le résultat de l'exercice 52 indiquant que $L \subseteq L'$ si et seulement si $L \cap \overline{L'} = \emptyset$ et la procédure de vérification associée sur les automates. Nous calculons donc deux automates, l'automate $A_1 \times A_2^c$ qui reconnaît $\mathcal{L}_{\text{auto}}(A_1) \cap \overline{\mathcal{L}_{\text{auto}}(A_2)}$ et l'automate $A_1^c \times A_2$ qui reconnaît $\overline{\mathcal{L}_{\text{auto}}(A_1)} \cap \mathcal{L}_{\text{auto}}(A_2)$. Ces deux automates sont les mêmes et sont représentés dans la figure 6.9e. D'après le théorème 5 et l'algorithme associé, ces deux automates reconnaissent le langage vide : aucun des états accessibles n'est accepteur.

Solution de l'exercice 63 (page 129)

1. Déterminons d'abord un automate pour L_1 en utilisant la construction de l'automate produit. Un AFED qui reconnaît les mots qui ne contiennent pas le facteur $abaa$ est représenté sous forme tabulaire dans la figure 6.10a. Un AFED qui reconnaît les mots qui contiennent un nombre pair d'occurrences du symbole a est représenté sous forme tabulaire dans la figure 6.10b. L'AFED produit des deux automates précédents est représenté dans la figure 6.10c.

Vérifions si ce dernier automate est minimal ou non. Selon le résultat produit par l'algorithme de minimisation (figure 6.10d), cet automate n'est pas minimal. Nous pouvons donc fusionner les états 40 et 41. L'automate minimisé où les états 40 et 41 ont été fusionnés est l'automate représenté dans la figure 6.10e (l'état 4 représente la classe groupant les états 40 et 41).

2. Déterminons d'abord un automate pour L_2 en utilisant la construction de l'automate produit. Un AFED qui reconnaît les mots qui ne contiennent pas le facteur aba est représenté dans la figure 6.11a. Un AFED qui reconnaît les mots qui contiennent un nombre d'occurrences du symbole a multiple de 4 est représenté dans la figure 6.11b. L'AFED produit des deux automates précédents est représenté dans la figure 6.11c. Vérifions si ce dernier automate est minimal ou non. Selon le résultat produit par l'algorithme de minimisation (figure 6.11d), cet automate n'est pas minimal. Nous pouvons donc fusionner les états 30, 31, 32 et 33. L'automate minimisé où les états 30, 31, 32 et 33 ont été fusionnés est l'automate représenté dans la figure 6.11e (l'état 3 représente la classe groupant les états 30, 31, 32 et 33).
3. Pour montrer que $L_2 \subseteq L_1$, nous construisons chacun des automates par composition et montrons $L_1 \cap L_2 = \emptyset$. L'AFED qui reconnaît $\overline{L_1}$ est représenté dans la figure 6.12a, obtenu en appliquant l'opération de complémentation sur l'automate qui reconnaît L_1 (figure 6.10e). L'AFED qui reconnaît L_2 est représenté dans la figure 6.12b (même automate que celui représenté dans la figure 6.11e). L'AFED produit est représenté dans la figure 6.12c. Comme on peut le voir sur cet automate, aucun des états accessibles n'est accepteur. Le langage reconnu par cet automate est donc vide.

	a	b
→ *0	1	0
*1	1	2
*2	3	0
*3	4	2
4	4	4

(a) AFED qui reconnaît les mots qui ne contiennent pas le facteur $abaa$.

	a	b
→ *0	1	0
	1	1

(b) AFED qui reconnaît les mots qui contiennent un nombre pair d'occurrences du symbole a .

	a	b
→ *00	11	00
*10	11	20
*20	31	00
*30	41	20
40	41	40
01	10	01
11	10	21
21	30	01
31	40	21
41	40	41

(c) AFED produit des AFED dans la figure 6.10a et la figure 6.10b

\equiv_0	\equiv_1	\equiv_2	\equiv_3	\equiv_4
00	00	00	00	00
10	10	10	10	10
20	20	20	20	20
30	30	30	30	30
01	01	01	01	01
11	11	11	11	11
21	21	21	21	21
31	31	31	31	31
40	40	40	40	40
41	41	41	41	41

(d) Minimisation de l'AFED produit dans la figure 6.10c.

	a	b
→ *00	11	00
*10	11	20
*20	31	00
*30	4	20
01	10	01
11	10	21
21	30	01
31	4	21
4	4	4

(e) Minimisation de l'AFED produit dans la figure 6.10c.

Figure 6.10 – Résultats pour l'exercice 63 (p. 129) - question 1.

	0	1
$\rightarrow *0$	1	0
*1	1	2
*2	3	0
3	3	3

(a) AFED qui reconnaît les mots qui ne contiennent pas le facteur aba .

	0	1
$\rightarrow *0$	1	0
1	2	1
2	3	2
3	0	3

(b) AFED qui reconnaît les mots qui contiennent un nombre d'occurrences du symbole a multiple de 4.

	0	1
$\rightarrow *00$	11	00
01	12	01
02	13	02
03	10	03
*10	11	20
11	12	21
12	13	22
13	10	23
*20	31	00
21	32	01
22	33	02
23	30	03
30	31	30
32	33	32
33	30	33
31	32	31

(c) AFED produit des AFED dans les figures 6.11a et 6.11b.

$\equiv 0$	$\equiv 1$	$\equiv 2$	$\equiv 3$	$\equiv 4$	$\equiv 5$
00	00	00	00	00	00
10	10	10	10	10	10
20	20	20	20	20	20
11	13	13	13	13	13
12	03	03	03	03	03
21	11	23	23	23	23
13	12	12	12	12	12
22	21	02	02	02	02
01	22	21	21	21	21
02	01	01	01	01	01
23	02	11	11	11	11
03	23	22	22	22	22
30	30	30	30	30	30
31	31	31	31	31	31
32	32	32	32	32	32
33	33	33	33	33	33

(d) Minimisation de l'AFED dans la figure 6.11c.

	0	1
$\rightarrow *00$	11	00
01	12	01
02	13	02
03	10	03
*10	11	20
11	12	21
12	13	22
13	10	23
*20	3	00
21	3	01
22	3	02
23	3	03
3	3	3

(e) AFED minimisé.

Figure 6.11 – Résultats pour l'exercice 63 (p. 129) - question 2.

	0	1
$\rightarrow 00$	11	00
10	11	20
20	31	00
30	4	20
*01	10	01
*11	10	21
*21	30	01
*31	4	21
*4	4	4

(a) Automate qui reconnaît $\overline{L_1}$.

	0	1
$\rightarrow *00$	11	00
01	12	01
02	13	02
03	10	03
*10	11	20
11	12	21
12	13	22
13	10	23
*20	3	00
21	3	01
22	3	02
23	3	03
3	3	3

(b) Automate qui reconnaît L_2 .

	$\downarrow 0000$	1111	1012	1113	1010	2121	2022	2123	2020	303
0	1111	1012	1113	1010	1111	303	313	303	313	43
1	0000	2121	2022	2123	2020	0101	0002	0103	0000	203

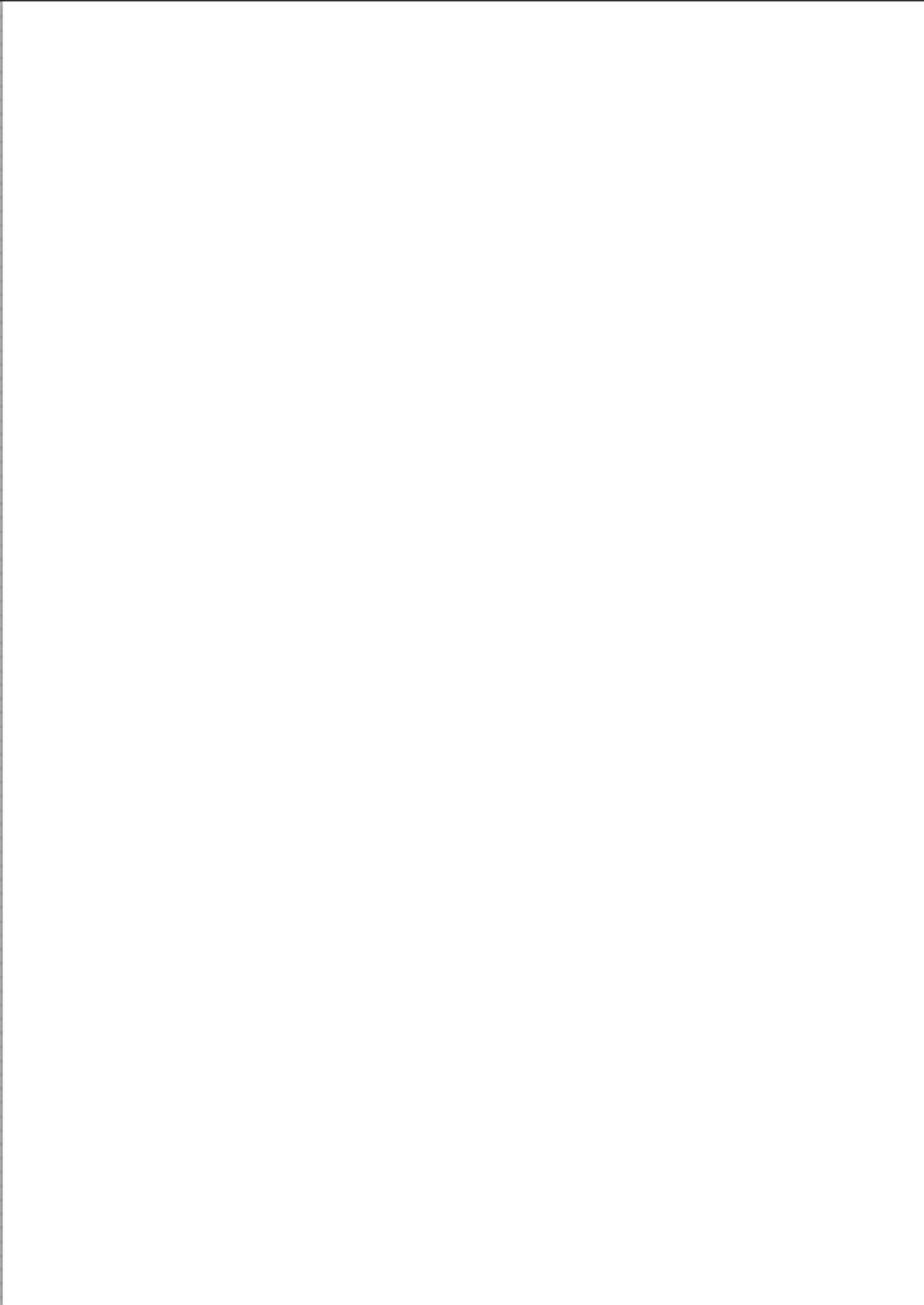
...

	0101	313	0002	0103	43	203	213	003	113	103
0	1012	43	1113	1010	43	313	303	113	103	113
1	0101	213	0002	0103	43	003	013	003	213	203

...

(c) Automate produit qui reconnaît $\overline{L_1} \cap L_2$.

Figure 6.12 – Résultats pour l'exercice 63 (p. 129) - question 3.



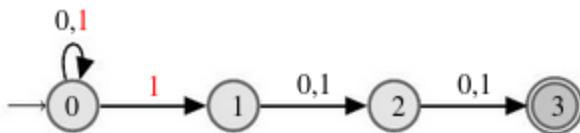
Chapitre 7

Automates non déterministes

7.1 Résumé intuitif du chapitre

Dans ce chapitre, nous nous intéressons à la notion d'**automate à nombre fini d'états et non déterministe**. La définition des automates à nombre fini d'états et non déterministes est plus souple sur la définition des transitions entre états de l'automate. Dans les automates à nombre fini d'états et déterministes introduits au chapitre 3, le mot déterministe réfère au fait que pour chaque état et pour chaque symbole de l'alphabet, il existe au plus un état successeur. Par opposition, dans les automates à nombre fini d'états et non déterministes, le mot non déterministe réfère au fait que pour un état et un symbole, il peut exister, zéro, une ou plusieurs états successeurs.

L'automate représenté ci-dessous est un automate à nombre fini d'états et non déterministe.



Cet automate est défini sur l'alphabet $\Sigma = \{0, 1\}$. Cet automate reconnaît le langage L_3 constitué des mots de longueur supérieure ou égale à 3 et dont le 3^{ème} symbole en partant de la droite est 1. Le non-déterminisme de l'automate est mis en exergue en rouge. À partir de l'état 0, deux transitions sont étiquetées par le symbole 1, une vers l'état 0 et une vers l'état 1.

Les motivations pour définir les automates non déterministes sont multiples. Tout d'abord, étant donné un langage L , il est souvent plus facile de trouver un automate non déterministe qui reconnaît ce langage.

De plus, pour certains langages, il est possible de trouver un automate non déterministe reconnaisseur qui est plus petit que tout automate déterministe reconnaisseur. Par exemple, pour le langage L_3 , le plus petit automate déterministe reconnaisseur a 8 états. Plus généralement, nous pourrons montrer que, si nous considérons le langage L_k constitué des mots de longueur supérieure à $k \in \mathbb{N}$ et dont le $k^{\text{ème}}$ symbole de droite est 1, aucun automate déterministe avec moins de 2^k états ne reconnaît L_k .

Toutefois, nous verrons qu'il n'est pas possible de se passer des automates déterministes. En effet, l'ensemble des opérations de composition vues au chapitre 4 ont été définies sur les automates à nombre fini d'états et déterministes. De plus, nous verrons que le non-déterminisme n'apporte pas d'expressivité dans la description des langages. Tous les langages que nous pouvons définir par un automate non déterministe peuvent être définis par un automate déterministe. Pour montrer cela, nous donnerons une procédure dite de **déterminisation**, prenant en entrée un automate à nombre fini d'états et non déterministe et produisant en sortie un automate à nombre fini d'états et déterministe qui reconnaît le même langage que l'automate à nombre fini d'états et non déterministe.

7.2 Les notions essentielles

7.2.1 Définition et langage reconnu

Définition 106 (Automate à nombre fini d'états et non déterministe, AFEND) *Un automate à nombre fini d'états et non déterministe, aussi appelé automate non déterministe et abrégé AFEND, est un quintuplet $(Q, \Sigma, q_{\text{init}}, \Delta, F)$ tel que :*

- Q est un ensemble fini d'états,
- Σ est l'alphabet de l'automate,
- $q_{\text{init}} \in Q$ est l'état initial,
- $\Delta \subseteq Q \times \Sigma \times Q$ est la relation de transition,
- $F \subseteq Q$ est l'ensemble des états accepteurs.

Remarque 29 Les définitions d'AFED (définition 63) et AFEND (définition 106) diffèrent uniquement par l'élément définissant les transitions : une fonction dans le cas des AFED et une relation dans le cas des AFEND. Toute fonction (définition 27 (p. 24)) étant une relation (définition 21 (p. 23)), la définition des AFEND est plus générale que celle des AFED. Plus précisément, tout AFED $(Q, \Sigma, q_{\text{init}}, \delta, F)$ est un AFEND $(Q, \Sigma, q_{\text{init}}, \Delta, F)$ avec $\Delta = \{(q, s, q') \mid \delta(q, s) = q'\}$.

Remarque 30 Nous représentons les AFEND de manière graphique ou tabulaire, de manière similaire aux AFED (remarque 65 (p. 46)); voir par exemple la figure 7.1 (p. 146) et le tableau 7.1 (p. 149).

Dans la suite, nous considérons un AFEND $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ et un mot $u \in \Sigma^*$ sur l'alphabet Σ .

Définition 107 (Configuration d'un AFEND) Une configuration de l'automate A est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.

Définition 108 (Relation de dérivation d'un AFEND) La relation de dérivation entre configurations, notée \rightarrow_Δ , est l'ensemble :

$$\{((q, a \cdot u), (q', u)) \mid q, q' \in Q \wedge a \in \Sigma \wedge u \in \Sigma^* \wedge (q, a, q') \in \Delta\}.$$

Lorsque $((q, a \cdot u), (q', u)) \in \rightarrow_\Delta$, on note $(q, a \cdot u) \rightarrow_\Delta (q', u)$.

Définition 109 (Exécution d'un AFEND) Une exécution de A sur un mot u à partir d'un état $q_1 \in Q$ est une séquence de configurations $(q_1, u_1) \cdots (q_n, u_n)$ telle que : $u_1 = u$ et $\forall i \in [1, n - 1], (q_i, u_i) \xrightarrow{\Delta} (q_{i+1}, u_{i+1})$.

Remarque 31 Il peut exister plusieurs exécutions d'un AFEND sur un mot.

Définition 110 (Acceptation d'un mot par un AFEND) Un mot $u \in \Sigma^*$ est accepté par A , s'il existe une exécution de A sur u (à partir de son état initial) $(q_{\text{init}}, u) \cdots (q_n, u_n)$ est telle que $q_n \in F$ et $u_n = \epsilon$. Dans le cas contraire, on dit que le mot est non accepté.

On dénote par \longrightarrow^*_Δ la fermeture réflexive et transitive de \longrightarrow_Δ .

Définition 111 (Langage reconnu/accepté par un AFEND) Le langage reconnu par A , noté $\mathcal{L}_{\text{auto}}(A)$, est l'ensemble

$$\{u \in \Sigma^* \mid u \text{ est accepté par } A\}.$$

7.2.2 Déterminisation des automates non déterministes

Définition 112 (Déterminisé d'un AFEND) Le déterminisé de A noté $\det(A)$, est l'AFED $(\mathcal{P}(Q), \Sigma, \{q_{\text{init}}\}, \delta, \mathcal{F})$ avec :

- $\mathcal{P}(Q)$ est l'ensemble des parties de Q , voir définition 16 (p. 22),
- $\delta(X, a) = \{q' \mid \exists q \in X, (q, a, q') \in \Delta\}$,
- $\mathcal{F} = \{X \in \mathcal{P}(Q) \mid X \cap F \neq \emptyset\}$ (c'est-à-dire, $X \in \mathcal{F}$ si et seulement si $X \cap F \neq \emptyset$).

Théorème 11 (Correction de la procédure de déterminisation) La procédure de déterminisation décrite par le déterminisé d'un automate (définition 112) est correcte, c'est-à-dire $\det(A)$ est un AFED et :

$$\mathcal{L}_{\text{auto}}(\det(A)) = \mathcal{L}_{\text{auto}}(A).$$

Un AFEND et son déterminisé reconnaissent donc le même langage, en ce sens ils sont équivalents. Par suite, la classe des langages reconnus par les automates non déterministes est la même que celle reconnue par les automates déterministes. Donc, en ce sens, les automates non déterministes sont équivalents aux automates déterministes.

Rappelons que pour une relation de transition (qui peut être une fonction) $\delta \subseteq Q \times \Sigma \times Q$, la fermeture réflexive et transitive de δ est dénotée par δ^* .

Définition 113 (Extension de la fonction de transition) Nous étendons la fonction de transition d'un AFED pour pouvoir représenter l'ensemble d'états atteints depuis un ensemble d'états :

- la fonction $\delta : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ représente l'ensemble d'états atteints à partir d'un symbole,
- la fonction $\delta^* : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$ représente l'ensemble d'états atteints à partir d'un mot.

Remarque 32 (À propos de la complexité de la déterminisation) En pratique, la taille du déterminisé (qui est un AFED) est sensiblement la même que celle de l'AFEND initial.

Nous décrivons un cas où la procédure de déterminisation « se passe mal ». Soit $\Sigma = \{a, b\}$ et soit L_k le langage constitué des mots de longueur supérieure ou égale à k et dont le $k^{\text{ème}}$ symbole de droite est 1 ; $L_k = \{s_1 \cdots s_n \mid n \geq k \wedge s_{n-k+1} = 1\}$. Le langage L_k est reconnu par l'AFEND représenté dans la figure 7.1. On vérifiera dans l'exercice 67 (p. 147) qu'aucun AFED avec moins de 2^k états ne reconnaît L_k .

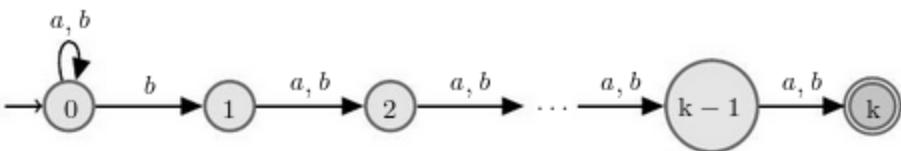


Figure 7.1 – AFEND qui reconnaît le langage L_k constitué des mots de longueur supérieure ou égale à k et dont le $k^{\text{ème}}$ symbole de droite est 1.

7.3 Exercices

7.3.1 Langage et AFEND

Exercice 64 (♣) — Mot accepté et non accepté par un AFEND

Nous considérons l'alphabet $\{a, b\}$ et l'AFEND dans la figure 7.2a.

1. Donner un mot accepté par cet AFEND et représenter ses exécutions sur le mot.
2. Donner un mot non accepté par cet AFEND et représenter ses exécutions sur le mot.

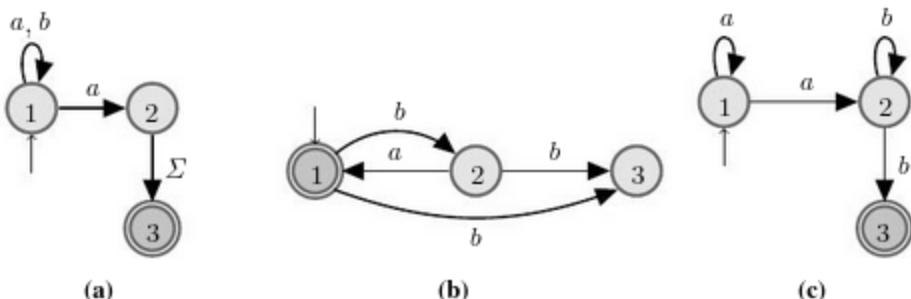


Figure 7.2 – Des automates non déterministes.

Exercice 65 (♣) — Langage reconnu par un AFEND

Nous considérons l'alphabet $\{a, b\}$.

1. Décrire en langage naturel chacun des automates représentés dans la figure 7.2.

Exercice 66 (♣) — AFEND qui reconnaît un langage avec une contrainte sur l'avant dernière lettre

Nous considérons l'alphabet $\{a, b\}$.

1. Donner un AFEND qui reconnaît tous les mots de longueur supérieure ou égale à 2 et tels que l'avant-dernier symbole est b.

Exercice 67 (♠) — AFEND qui reconnaît un langage avec une contrainte sur un symbole de la fin des mots

Nous considérons l'alphabet $\Sigma = \{a, b\}$. Nous définissons le langage L_i comme l'ensemble des mots de Σ^* tels que tous les symboles à partir du i -ème symbole et jusqu'à la fin du mot est le symbole a . (Le premier symbole d'un mot en partant de la fin est le dernier symbole du mot.)

1. Définir formellement le langage L_i , pour $i \in \mathbb{N}$.
2. Donner un AFEND qui reconnaît L_1 .
3. Donner un AFEND qui reconnaît L_2 .
4. Donner un AFEND qui reconnaît L_3 .

Les automates donnés doivent être non déterministes.

Exercice 68 (♠♠♣) — Un AFEND qui reconnaît le langage des longueurs

Soit A un AFEND reconnaissant un langage L . Nous considérons le langage :

$$\text{longueur}(L) = \{u \in \Sigma^* \mid \exists v \in L. |u| = |v|\}.$$

1. À partir de A , donner un AFEND qui reconnaît $\text{longueur}(L)$.
2. Démontrer que le langage reconnu par cet AFEND est bien le langage attendu.

Exercice 69 (♠♠♣♣) — Donner un AFEND qui reconnaît les solutions d'équations

Pour $n \in \mathbb{N}$, soit \widehat{n} l'ensemble des représentations en binaire de n où le bit de poids le plus faible est à gauche.

Par exemple :

- Pour $6 \in \mathbb{N}$: $011 \in \widehat{6}$ et $0110 \in \widehat{6}$.
- Pour $2 \in \mathbb{N}$: $010 \in \widehat{2}$ et $01 \in \widehat{2}$.

Considérons l'alphabet $\Sigma = \{0, 1\} \times \{0, 1\}$, nous cherchons un AFEND qui reconnaît $L \subseteq \Sigma^*$ tel que $u \in L$ si et seulement s'il existe $x, y \in \mathbb{N}$ tels que :

- $u = (x_1, y_1) \cdots (x_k, y_k)$,
- $x_1 \cdots x_k \in \widehat{x}$,
- $y_1 \cdots y_k \in \widehat{y}$, et
- $y = 2 \times x$.

1. Donner une définition en extension de l'alphabet de cet automate.
2. Donner une condition nécessaire et suffisante pour que deux représentations binaires d'entiers naturels représentent des entiers tels que $y = 2 \times x$.
3. Observer que l'alphabet de l'automate fait que les représentations binaires lues en entrées sont de la même longueur. Revisiter la condition de la question précédente.
4. Étant donné un mot lu sur l'alphabet, nous distinguons les trois situations suivantes :
 - (a) Le mot lu jusqu'à présent représente une solution de l'équation.
 - (b) Le mot lu jusqu'à présent ne représente pas une solution de l'équation mais peut représenter une solution en lisant un certain symbole.
 - (c) Le mot lu jusqu'à présent ne représente pas une solution de l'équation et aucune extension de ce mot ne pourra former de solution de l'équation.

Pour chacune de ces situations, nous associons un état de l'automate. Pour chacun de ces états, indiquer comment la lecture de chaque symbole de l'alphabet fait changer d'état. Ceci permet de déduire la fonction de transition de l'automate.

5. Déterminer l'état initial et les états accepteurs, puis donner un AFEND qui reconnaît les solutions de $y = 2 \times x$ dans \mathbb{N} .

Exercice 70 (♠♠♠) — Automates non déterministes généralisés

La notion d'AFEND peut être généralisée en considérant qu'un automate possède un ensemble non vide d'états initiaux. Nous souhaitons montrer que ce type d'automate est équivalent aux AFEND classiques.

1. Adapter les notions de configuration, d'exécution, et de langage reconnu par ce type d'automates.
2. Indiquer comment montrer que ces automates sont équivalents aux AFEND.

7.3.2 Déterminiser un AFEND

Exercice 71 (♠♠) — Déterminiser en utilisant la représentation graphique

Soit A l'AFEND défini par $(\{1, 2, 3, 4, 5, 6\}, \{a, b\}, 1, \Delta, \{2\})$ tel que :

$$\Delta = \{(1, a, 2), (2, a, 3), (3, a, 2), (2, a, 4), (4, b, 2), (2, b, 5), (5, a, 2), (2, b, 6), (6, b, 2)\}.$$

1. Représenter A graphiquement.
2. En utilisant la représentation graphique, déterminiser A .
3. Minimiser l'AFED obtenu à la question précédente.

Exercice 72 (♠♠) — Déterminiser en utilisant la représentation tabulaire

Considérons les deux automates suivants :

- A , l'AFEND défini par $(\{0, 1, 2, 3, 4, 5\}, \{a, b\}, 0, \Delta, \{4\})$ dont la relation de transition est définie par le tableau 7.1a.

Tableau 7.1 – Des automates non déterministes représentés sous représentation tabulaire.

	$\downarrow 0$	1	2	3	4^*	5
a	1, 2, 3, 4, 5	2, 3	0, 1, 4	0	1	2
b		4	1, 2, 3	1, 2, 5		2, 3, 5

(a)

	$\downarrow 0^*$	1	2	3^*	4^*	5
a	1, 2			5		
b		3	4			0

(b)

- B , l'AFEND donné par $(\{0, 1, 2, 3, 4, 5\}, \{a, b\}, 0, \Delta, \{0, 3, 4\})$ dont la relation de transition est définie par le tableau 7.1b.
1. En utilisant sa représentation graphique, déterminiser A et minimiser l'AFED obtenu après déterminisation.
 2. Même question pour l'automate B .

Exercice 73 (♠♦) — Déterminiser un AFEND

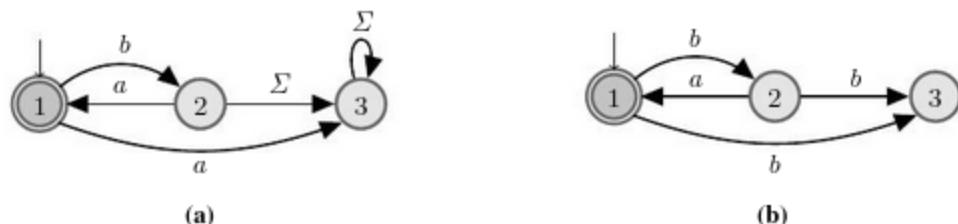
Considérons les AFEND représentés dans la figure 7.2.

1. Pour chacun de ces automates, construire AFED équivalent par déterminisation en utilisant la représentation d'automates de votre choix.

Exercice 74 (♠♦) — Déterminiser un AFEND

Considérons les AFEND représentés dans la figure 7.3.

1. Pour chacun de ces automates, construire AFED équivalent par déterminisation en utilisant la représentation d'automates de votre choix.

**Figure 7.3** – Des automates non déterministes.**7.3.3 Équivalence entre AFEND****Exercice 75 (♣) — Procédures pour déterminer l'équivalence**

Nous considérons les procédures/algorithmes utilisés dans les chapitres précédents pour déterminer l'équivalence entre deux AFED.

1. Adapter les procédures/algorithmes aux AFEND.

Exercice 76 (♠♦) — Déterminer l'équivalence ou la non-équivalence entre deux AFEND

Soit $\Sigma = \{0, 1\}$. Considérons les deux AFEND représentés dans la figure 7.3.

1. Quel est le langage reconnu par l'automate représenté dans la figure 7.3a ?
2. Quel est le langage reconnu par l'automate représenté dans la figure 7.3b ?
3. Montrer que ces deux automates sont équivalents.

7.3.4 Complexité

Exercice 77 (♠♠♣) — Nombre d'exécutions acceptées

Par abus de langage, nous appelons exécution acceptée, une exécution dont la dernière configuration est acceptante. Nous nous intéressons à calculer le nombre $\mathcal{N}(A, u)$ d'exécutions acceptées par un automate A pour un mot u donné en entrée. Rappelons que $|u|_a$ dénote le nombre d'occurrences du symbole a dans le mot u .

1. Considérons l'automate A_1 représenté dans la figure 7.4a, lister les exécutions acceptées et associées au mot $abaa$. En déduire $\mathcal{N}(A_1, abaa)$.
2. Montrer que $\mathcal{N}(A_1, u) = |u|_a$.
3. Considérons l'automate A_2 représenté dans la figure 7.4b, quelle est la valeur de $\mathcal{N}(A_2, u)$ pour un mot u . Justifier.
4. Considérons l'automate A_3 représenté dans la figure 7.4c, quelle est la valeur de $\mathcal{N}(A_3, u)$ pour un mot u . Justifier.
5. Considérons deux AFEND $Auto_1$ et $Auto_2$ sur l'alphabet Σ et l'AFEND $Auto$ résultant du produit pour intersection de $Auto_1$ et $Auto_2$.
Montrer que $\forall u \in \Sigma^*, \mathcal{N}(Auto, u) = \mathcal{N}(Auto_1, u) \times \mathcal{N}(Auto_2, u)$.
6. Donner un automate A tel que $\forall u \in \Sigma^*, \mathcal{N}(A, u) = (|u|_a)^2$.

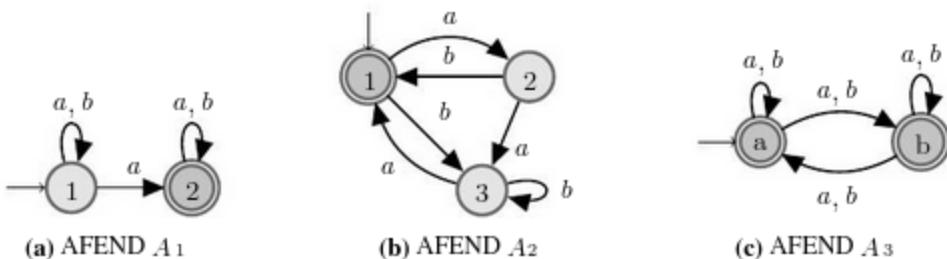


Figure 7.4 – Des automates non déterministes

Exercice 78 (♠♣♣) — Complexité de la déterminisation avec le nombre d'états, exemples

Nous considérons les langages L_i tels que définis dans l'exercice 67 (p. 147).

1. Donner le nombre d'états des AFED reconnaissant L_1 , L_2 et L_3 . Vous pouvez utiliser un outil tel que Aude pour trouver l'AFED qui reconnaît L_3 .
2. Que pouvons-nous extrapoler pour le nombre d'états d'un AFED reconnaissant L_n pour $n \in \mathbb{N}$?

Exercice 79 (♠♦♦) — Complexité de la déterminisation avec le nombre d'états, généralisation

Nous considérons l'algorithme de déterminisation vu en cours opérant par construction des sous-ensembles.

1. Au pire des cas, pour un AFEND avec n états, quel est le nombre d'états de l'AFED équivalent obtenu par cet algorithme c'est-à-dire en suivant la définition du déterminisé ?
2. En s'inspirant de l'algorithme de parcours des états d'un automate et de l'algorithme de calcul du produit de deux automates « à la volée », écrire un algorithme de déterminisation « à la volée » qui produit le déterminisé d'un AFEND.
3. Déterminer une condition sur l'exécution de l'algorithme pour que le pire cas se produise.
4. Déterminer le nombre de fois où chaque opération de l'algorithme s'exécute dans le pire cas.
5. En supposant un temps d'exécution moyen de 1ms pour chaque opération de l'algorithme, donner une estimation du temps d'exécution de l'algorithme de déterminisation lorsqu'il est exécuté sur des AFEND de différentes tailles.

7.3.5 Algorithmes pour les AFEND

Exercice 80 (♠♦) — Algorithmes pour décider le déterminisme

Considérons deux alphabets Σ et Σ' tels que $\Sigma' \subseteq \Sigma$ et un AFEND défini sur Σ . Pour cet exercice, nous définissons la notion de déterminisme par rapport à un alphabet. Nous disons qu'un AFEND défini sur l'alphabet Σ est déterministe par rapport à Σ' si sa relation de transition est une fonction lorsqu'elle est restreinte à Σ' .

1. Donner un algorithme qui détermine si un AFEND défini sur Σ est déterministe par rapport à Σ' .
2. En déduire un algorithme qui détermine si un AFEND est un AFED.

7.4 Indications pour résoudre les exercices

Indications pour l'exercice 68 (p. 147)

1. Partir de l'automate A reconnaissant L et augmenter les transitions : quand il existe une transition entre deux états sur un certain symbole, ajouter des transitions entre ces

deux états sur chaque symbole de l'alphabet. Ainsi, s'il existe un mot depuis l'état initial vers un état accepteur, tous les mots de même longueur seront acceptés par l'automate.

2. *Définir l'ensemble $T(n, \Delta)$ des triplets (q, u, q') tels que l'exécution de l'automate sur le mot u à partir de l'état q atteint l'état q' , en utilisant la relation de transition Δ . Démontrer la propriété suivante par récurrence sur $n \in \mathbb{N}^*$:*

$$\forall n \in \mathbb{N}^*. \forall u \in \Sigma^n. ((q, u, q') \in T(n, \Delta')) \iff \exists v \in \Sigma^n. (q, v, q') \in T(n, \Delta).$$

Montrer ensuite que $\text{longueur}(A)$ reconnaît $\text{longueur}(L)$ en utilisant la propriété démontrée.

Indications pour l'exercice 69 (p. 147)

2. *Pour que deux représentations binaires $x_1 \cdot x_2 \cdots x_{k_x}$ et $y_1 \cdot y_2 \cdots y_{k_y}$ représentent deux entiers naturels x et y tels que $y = 2 \times x$, il faut que $k_y = k_x + 1$, $y_1 = 0$ et $x_i = y_{i+1}$.*
3. *Distinguer des situations en fonction du fait que le mot lu jusqu'à présent représente une solution de l'équation ou pas. Déterminer comment la lecture de chaque symbole de l'alphabet fait évoluer ces situations.*
4. *Les situations distinguées à la question précédente déterminent les états de l'automate. Déterminer l'état initial.*

Indications pour l'exercice 70 (p. 148)

1. *Peu de choses changent par rapport aux AFEND. La différence principale est que, lors de la lecture d'un mot, les exécutions seront « générées » à partir de chacun des états initiaux.*
2. *Montrer qu'un automate généralisé peut se traduire en AFEND équivalent, et vice-versa.*

Indications pour l'exercice 76 (p. 150)

3. *Utiliser l'une des procédures pour montrer l'équivalence entre automates, section 6.2.4 (p. 125). Par exemple, utiliser la procédure basée sur la minimisation : deux automates sont équivalents suivant cette procédure si et seulement si leur deux automates minimisés sont égaux modulo un renommage des états.*

Indications pour l'exercice 77 (p. 150)

2. *Démontrer la propriété par induction sur $u \in \{a, b\}^*$. Distinguer les exécutions qui terminent dans l'état 1 de celles qui terminent dans l'état 2.*

Indications pour l'exercice 78 (p. 150)

2. *Le nombre d'états croit exponentiellement en fonction de n .*

Indications pour l'exercice 79 (p. 151)

1. Raisonnner en termes d'accessibilité.
2. Adapter l'algorithme de parcours pour réaliser un parcours où l'on construit les états du déterminisé.
3. Dans l'algorithme trouvé à la question précédente, adapté de l'algorithme de parcours, examiner la condition pour que les itérations se poursuivent.

Indications pour l'exercice 80 (p. 151)

1. Parcourir tous les états et tous les symboles de Σ' .

7.5 Solutions des exercices

7.5.1 Langage et AFEND

Solution de l'exercice 64 (page 146)

L'exécution d'un mot est l'ensemble des séquences de configurations possibles liées à ce mot. Nous représentons cet ensemble de séquences sous forme arborescente pour représenter « les choix faits par l'automate » liés à l'aspect non déterministe.

1. Le mot *baa* est accepté par cet automate. L'exécution de l'automate sur *baa* est représentée sur la figure 7.5a. Nous observons que l'une des exécutions de l'automate termine dans la configuration acceptante $(3, \epsilon)$.
2. Le mot *aba* est rejeté par cet automate. L'exécution de l'automate sur *aba* est représentée sur la figure 7.5b. Nous observons qu'aucune des exécutions de l'automate termine dans une configuration acceptante.

Solution de l'exercice 65 (page 146)

1. L'ensemble des mots sur l'alphabet tels que l'avant dernier symbole est *a*.
2. L'ensemble des mots formés par une répétition du facteur $b \cdot a$.
3. L'ensemble des mots qui commencent par des *a* et qui finissent par des *b*.

Solution de l'exercice 66 (page 147)

1. L'automate est représenté dans la figure 7.6a.

Solution de l'exercice 67 (page 147)

1. Le langage L_i peut être défini par :

$$\{s_1 \cdot s_2 \cdots s_{n-i+1} \cdots s_n \mid n \geq i \wedge s_{n-i+1} = a \wedge \forall j \in [1, n] . s_j \in \{a, b\}\}.$$

Ou alternativement par :

$$\{s_n \cdot s_{n-1} \cdots s_i \cdots s_1 \mid n \geq i \wedge s_i = a \wedge \forall j \in [1, n] . s_j \in \{a, b\}\}.$$

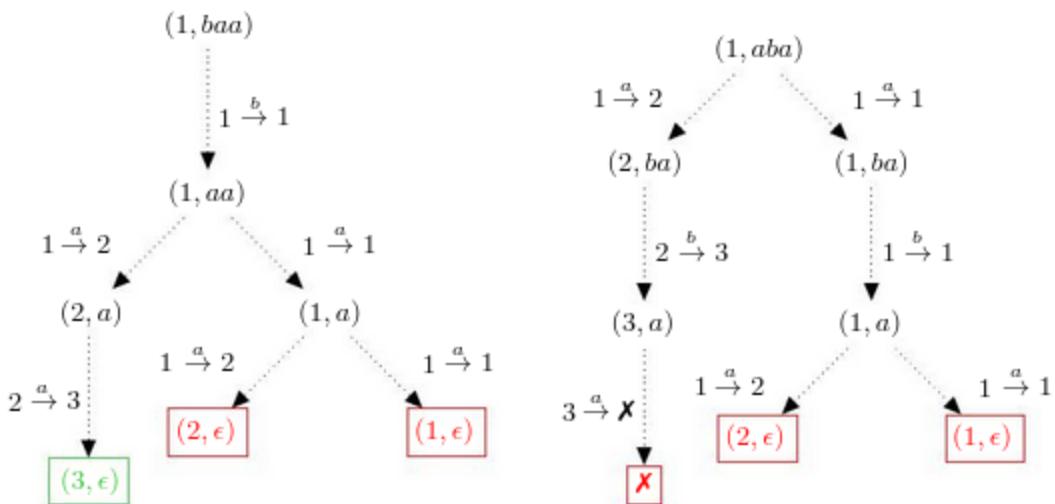


Figure 7.5 – Exécutions de l’automate de la figure 7.2a sur deux mots : *baa* (figure 7.5a) et *aba* (figure 7.5b). Chaque exécution est représentée sous forme d’arbre dont les nœuds sont des configurations, les feuilles (encadrées) des configurations terminales, en vert les configurations acceptantes, en rouge les configurations non acceptantes. Chaque arête est étiquetée par la transition justifiant l’évolution de la configuration correspondante.

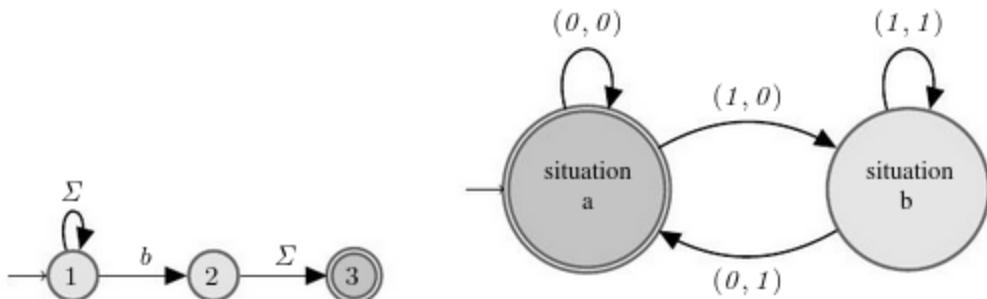
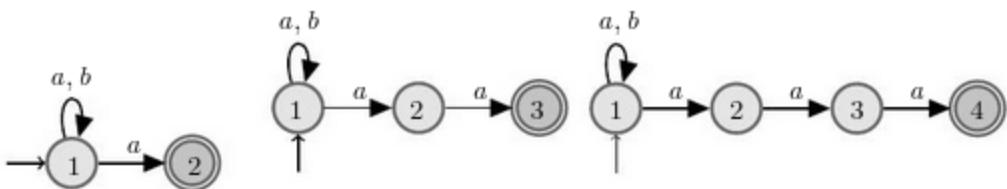


Figure 7.6 – Automates solutions de plusieurs exercices.

2. L’AFEND qui reconnaît L_1 est représenté dans la figure 7.7a.
3. L’AFEND qui reconnaît L_2 est représenté dans la figure 7.7b.
4. L’AFEND qui reconnaît L_3 est représenté dans la figure 7.7c.

Solution de l’exercice 68 (page 147)

Pour les deux questions suivantes, nous considérons un AFEND $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ qui reconnaît L .

(a) AFEND qui reconnaît L_1 . (b) AFEND qui reconnaît L_2 . (c) AFEND qui reconnaît L_3 .**Figure 7.7** – Résultats pour l'exercice 67 (p. 147).

De plus, nous utilisons des ensembles de triplets de la forme (q, u, q') , formés par un état q de Q , un mot u et un état q' de Q , tels que l'exécution de l'automate sur le mot u à partir de l'état q atteint l'état q' . Ces ensembles sont paramétrés par un entier donnant la longueur des mots considérés et par une relation de transition $\Delta \subseteq Q \times \Sigma \times Q$. Ces ensembles sont définis comme suit :

- $T(1, \Delta) = \Delta$, et
- $T(n + 1, \Delta) = \{(q, u' \cdot a, q') \in Q \times \Sigma^{n+1} \times Q \mid \exists q'' \in Q. (q, u', q'') \in T(n, \Delta) \wedge (q'', a, q') \in \Delta\}$, pour $n > 1$.

$T(n, \Delta)$ est l'ensemble des triplets (q, u, q') tels que l'exécution de l'automate sur le mot u à partir de l'état q atteint l'état q' , en utilisant la relation de transition Δ . Notons que pour $n \in \mathbb{N}^*$, par construction, les mots apparaissant dans les triplets de $T(n, \Delta)$ sont de longueur n .

En conséquence du critère d'acceptation des AFEND, un mot u de longueur n est accepté par A si et seulement si $(q_{\text{init}}, u, q_f) \in T(n, \Delta)$, avec $q_f \in F$.

1. Nous définissons l'AFEND longueur(A) = $(Q, \Sigma, q_{\text{init}}, \Delta', F)$, avec :

$$\Delta' = \bigcup_{(q, a, q') \in \Delta} \{q\} \times \Sigma \times \{q'\}$$

Pour chaque transition (q, a, q') de Δ , Δ' contient les transitions depuis l'état q vers l'état q' sur chaque symbole de Σ .

2. Pour démontrer que le langage reconnu par cet automate est bien le langage attendu, nous utilisons la propriété suivante :

$$\forall n \in \mathbb{N}^*. \forall u \in \Sigma^n. ((q, u, q') \in T(n, \Delta') \iff \exists v \in \Sigma^n. (q, v, q') \in T(n, \Delta)).$$

Démontrons la propriété par récurrence sur $n \in \mathbb{N}^*$.

- Cas de base : $n = 1$.

La propriété est vérifiée par définition de Δ' et $T(n, \Delta')$.

- Pas d'induction.

Soit $n \geq 1$, supposons la propriété vérifiée pour n . Considérons $(q, u \cdot a, q') \in T(n + 1, \Delta')$, alors il existe $q_1 \in Q$ tel que $(q, u, q_1) \in T(n, \Delta')$ et $(q_1, a, q') \in T(1, \Delta')$. En utilisant $(q, u, q_1) \in T(n, \Delta')$, ainsi que l'hypothèse d'induction, nous en déduisons que $\exists v \in \Sigma^n. (q, v, q_1) \in T(n, \Delta)$. En utilisant

$(q_1, a, q') \in T(1, \Delta')$, nous déduisons $\exists a' \in \Sigma. (q_1, a', q') \in \Delta$. En utilisant $(q, v, q_1) \in T(n, \Delta)$ et $(q_1, a', q') \in \Delta$, nous déduisons $(q, v \cdot a', q') \in T(n+1, \Delta)$.

Le même raisonnement peut être suivi dans l'autre sens pour montrer la réciproque.

Montrons maintenant que $\text{longueur}(A)$ reconnaît $\text{longueur}(L)$.

- Considérons $u \in \Sigma^*$ un mot de longueur $n \in \mathbb{N}^*$ accepté par $\text{longueur}(A)$. En utilisant l'exécution de $\text{longueur}(A)$ sur u , nous trouvons $(q_{\text{init}}, u, q_f) \in T(n, \Delta')$ avec $q_f \in F$. En appliquant la propriété précédente, nous trouvons $\exists v \in \Sigma^n. (q_{\text{init}}, v, q_f) \in T(n, \Delta)$. C'est-à-dire qu'il existe un mot v tel que $|u| = |v|$ et v soit accepté par A , c'est-à-dire, $v \in L$. Par définition de $\text{longueur}(L)$, $u \in \text{longueur}(L)$.
- Considérons $u \in \text{longueur}(L)$, un mot de longueur $n \in \mathbb{N}^*$. Comme $u \in \text{longueur}(L)$, il existe $v \in \Sigma^*$ tel que $|v| = |u|$ et $v \in L$, v est un mot accepté par A . Comme v est accepté par A , en utilisant son exécution, nous trouvons $(q_{\text{init}}, v, q_f) \in T(n, \Delta)$. En utilisant la propriété prouvée précédemment, nous trouvons que pour tous les mots de longueur n et pour u en particulier, nous avons $(q_{\text{init}}, u, q_f) \in T(n, \Delta')$. De manière similaire, cela veut dire que u est accepté par $\text{longueur}(A)$.

Solution de l'exercice 69 (page 147)

1. L'alphabet $\{0, 1\} \times \{0, 1\}$ est égal à $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$.
2. Pour que deux représentations binaires $x_1 \cdot x_2 \cdots x_{k_x}$ et $y_1 \cdot y_2 \cdots y_{k_y}$ représentent deux entiers naturels x et y tels que $y = 2 \times x$, il faut que $k_y = k_x + 1$, $y_1 = 0$ et $x_i = y_{i+1}$. C'est-à-dire que la représentation de y commence par un 0 ($y_1 = 0$) et est ensuite égale à la représentation binaire de x pour les symboles suivants, dans le même ordre.
3. Pour que deux mots de même longueur $x_1 \cdots x_k$ et $y_1 \cdots y_k$ encodant des entiers x et y soient tels que $y = 2 \times x$, il faut donc que :
 - $y_1 = 0$,
 - $x_k = 0$ et
 - $x_i = y_{i+1}$, pour $i \in [1, k - 1]$.
4. En se basant sur la condition trouvée à la question précédente, nous en déduisons l'évolution des situations sur chaque symbole de l'alphabet.
 - (a) Dans la situation où le mot d'entrée représente des entiers naturels qui sont solutions de l'équation, les symboles de l'alphabet font évoluer la situation comme suit :
 - La lecture du symbole $(0, 0)$ nous laisse dans la même situation, car l'ajout du symbole 0 dans cette représentation d'entiers naturels ne change pas leur valeur.
 - La lecture du symbole $(1, 0)$ fait passer dans la situation (b) et le prochain symbole du mot représentant y doit être un 1 et celui de x un 0 pour revenir

- à la situation (a), rétablissant ainsi le fait que le mot y est le mot x décalé d'un symbole sur la droite.
- La lecture des autres symboles ((0, 1) et (1, 1)) fait passer dans la situation (c) car les mots représentant x et y ne pourront plus satisfaire la condition.
- (b) Dans la situation où le mot lu jusqu'à présent ne représente pas une solution de l'équation mais peut représenter une solution en lisant certains symboles, les symboles de l'alphabet font évoluer la situation comme suit :
- La lecture du symbole (1, 1) nous laisse dans la même situation, car l'ajout du symbole 1 aux deux nombre dans cette représentation d'entiers naturels nous laisse dans la situation où le mot y est le mot x décalé d'un symbole sur la droite si le prochain symbole reçu pour x est 0 et celui pour y un 1.
- (c) Dans la situation où le mot lu jusqu'à présent ne représente pas une solution de l'équation et aucune extension de ce mot ne pourra former de solution de l'équation, la lecture de n'importe quel symbole nous laisse dans la même situation.
5. Initialement, les deux mots lus en entrées sont vides, ils représentent tous deux l'entier naturel 0, qui vérifient l'équation. Il y a un seul état accepteur qui correspond à la première situation (et qui est l'état initial). L'automate est donné dans la figure 7.6b (p. 154). Notons que l'AFEND n'est pas complet, en particulier, il n'y a pas d'état lié à la situation c, et les transitions menant à cet état ne sont pas représentées.
- Solution de l'exercice 70 (page 148)**
1. Un automate généralisé par un ensemble d'états initiaux peut être défini par un quintuplet $(Q, \Sigma, Q_{\text{init}}, \Delta, F)$ où $Q_{\text{init}} \subseteq Q$ et les autres éléments sont comme dans la définition des AFEND.
 - Une configuration d'un automate généralisé est un élément de $Q \times \Sigma^*$ et a la même définition que celle des AFEND : un couple formé d'un état et d'un mot.
 - Une exécution d'un automate généralisé sur un mot u est une séquence de configurations $(q_0, u_0) \cdot (q_1, u_1) \cdots (q_n, u_n)$ telle que $q_0 \in Q_{\text{init}}$, $u_n = \epsilon$ et $\forall i \in [0, n - 1]. (q_i, u_i) \xrightarrow{\Delta} (q_{i+1}, u_{i+1})$; c'est une définition similaire à celle des AFEND, à l'exception que la séquence de configuration peut commencer dans l'un des états de l'ensemble Q_{init} .
 - Un mot est accepté par un automate généralisé à la même condition que pour un AFEND : il doit avoir (au moins) une exécution acceptée par l'automate.
 - Le langage accepté par un automate généralisé a la même définition que pour un AFEND : c'est l'ensemble des mots acceptés.
 2. Pour montrer que ces automates sont équivalents aux AFEND, il faut montrer que l'on peut traduire un automate généralisé vers un AFEND équivalent (c'est-à-dire qui reconnaît le même langage), et vice-versa.
 - La traduction d'AFEND vers automate généralisé est évidente.
 - Pour définir la traduction d'un automate généralisé vers un AFEND, nous considérons un automate généralisé $(Q, \Sigma, Q_{\text{init}}, \Delta, F)$ et l'ensemble des transitions

sortantes des états initiaux de l'automate généralisé $\Delta_{\text{init}} = \{(q_{\text{init}}, s, q) \in \Delta \mid q_{\text{init}} \in Q_{\text{init}} \wedge s \in \Sigma \wedge q \in Q\}$, créons un nouvel état q'_{init} ($q'_{\text{init}} \notin Q$) qui devient l'état initial dans l'automate traduit, et nous ajoutons à Δ l'ensemble des transitions dans Δ_{init} pour lesquelles nous choisissons pour l'état initial q'_{init} , c'est-à-dire nous ajoutons $\{(q'_{\text{init}}, s, q) \mid \exists (q_{\text{init}}, s, q) \in \Delta_{\text{init}}\}$. L'automate résultat est :

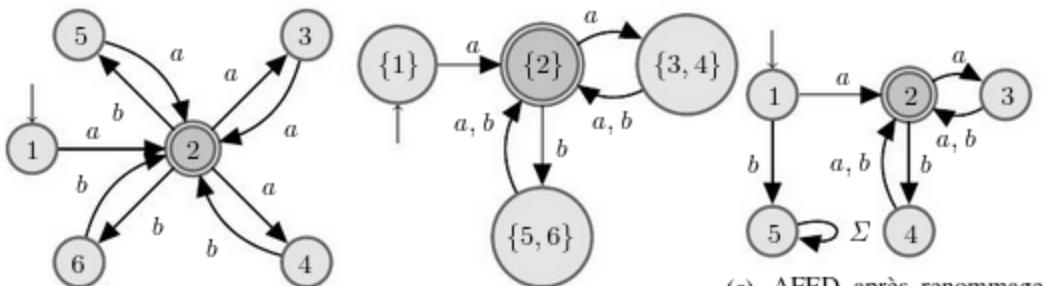
$$(Q \cup \{q'_{\text{init}}\}, \Sigma, q'_{\text{init}}, \Delta \cup \{(q'_{\text{init}}, s, q) \mid \exists (q_{\text{init}}, s, q) \in \Delta_{\text{init}}\}, F).$$

7.5.2 Déterminiser un AFEND

Solution de l'exercice 71 (page 148)

1. L'AFEND A est représenté graphiquement dans la figure 7.8a.
2. Nous déterminisons cet automate, c'est-à-dire nous calculons $\text{det}(A)$. Le résultat est représenté dans la figure 7.8b.

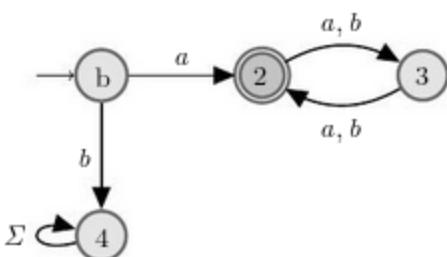
Nous détaillons l'obtention de $\text{det}(A)$ en réalisant une *construction paresseuse* par exploration des états accessibles depuis l'état initial, c'est-à-dire que l'on ne calcule que les successeurs des états déjà atteints. Dans l'exercice 79, vous devrez trouver l'algorithme réalisant cette déterminisation.



(a) AFEND A en représentation (b) AFED déterminisé de l'AFEND (c) AFED après renommage
graphique. des états de l'AFED dans la figure 7.8a. des états de l'AFED dans la figure 7.8b.

\equiv_0	\equiv_1	\equiv_2
1	1	1
4	4	4
3	3	3
5	5	5
2	2	2

(d) Minimisation de l'AFED dans la figure 7.8c.



(e) Minimisé de l'AFED dans la figure 7.8c.

Figure 7.8 – Résultats pour l'exercice 71 (p. 148).

- L'état initial de $\text{det}(A)$ est l'ensemble singleton formé par l'état initial de A , c'est-à-dire $\{1\}$.
- À partir de l'état $\{1\}$, nous examinons quelles transitions sont possibles, en examinant chaque symbole de l'alphabet. Sur le symbole a , dans l'automate A , on atteint l'état 2 uniquement. Il y a donc une transition depuis $\{1\}$ vers $\{2\}$ sur le symbole a dans $\text{det}(A)$. Sur le symbole b , dans l'automate A , il n'y a pas de transition définie. Il n'y a donc pas de transition depuis $\{1\}$ sur le symbole b dans $\text{det}(A)$. (Si nous souhaitions que $\text{det}(A)$ soit complet, nous aurions pu ajouter une transition depuis $\{1\}$ vers un état puits \emptyset .) L'exploration des états à partir de $\{1\}$ a produit l'état $\{2\}$ que nous devons explorer.
- À partir de l'état $\{2\}$, nous examinons quelles transitions sont possibles, en examinant chaque symbole de l'alphabet. Sur le symbole a , dans l'automate A , on atteint les états 3 et 4. Il y a donc une transition depuis $\{2\}$ vers $\{3, 4\}$ sur le symbole a dans $\text{det}(A)$. Sur le symbole b , dans l'automate A , on atteint les états 5 et 6. Il y a donc une transition depuis $\{2\}$ vers $\{5, 6\}$ sur le symbole b dans $\text{det}(A)$. L'exploration des états à partir de $\{2\}$ a produit les états $\{3, 4\}$ et $\{5, 6\}$ que nous devons explorer.
- À partir de l'état $\{3, 4\}$, nous examinons quelles transitions sont possibles, en examinant chaque symbole de l'alphabet. Sur le symbole a , dans l'automate A , on atteint l'état 2 à partir de l'état 3 et aucun état à partir de l'état 4. Il y a donc une transition depuis $\{3, 4\}$ vers $\{2\}$ sur le symbole a dans $\text{det}(A)$. Sur le symbole b , dans l'automate A , on atteint l'état 2 à partir de l'état 4 et aucun état à partir de l'état 3. Il y a donc une transition depuis $\{3, 4\}$ vers $\{2\}$ sur le symbole b dans $\text{det}(A)$. L'exploration des états à partir de $\{3, 4\}$ a produit aucun nouvel état.
- À partir de l'état $\{5, 6\}$, nous examinons quelles transitions sont possibles, en examinant chaque symbole de l'alphabet. Sur le symbole a , dans l'automate A , on atteint l'état 2 à partir de l'état 5 et aucun état à partir de l'état 6. Il y a donc une transition depuis $\{5, 6\}$ vers $\{2\}$ sur le symbole a dans $\text{det}(A)$. Sur le symbole b , dans l'automate A , on atteint l'état 2 à partir de l'état 6 et aucun état à partir de l'état 5. Il y a donc une transition depuis $\{5, 6\}$ vers $\{2\}$ sur le symbole b dans $\text{det}(A)$. L'exploration des états à partir de $\{3, 4\}$ a produit aucun nouvel état.
- L'exploration des états se terminent car plus aucun nouvel état n'est découvert.
- Nous devons déterminer les états accepteurs de $\text{det}(A)$. Ces états sont ceux qui contiennent un état accepteur de A . Ainsi, l'unique état accepteur de $\text{det}(A)$ est l'état $\{2\}$.

Ensuite, nous renommons les états et complétons l'automate. Le résultat est représenté dans la figure 7.8c.

3. Nous minimisons l'automate en appliquant l'algorithme de minimisation (dont le résultat est représenté dans la figure 7.8d) et représentons l'automate minimal dans la figure 7.8e.

Solution de l'exercice 72 (page 148)

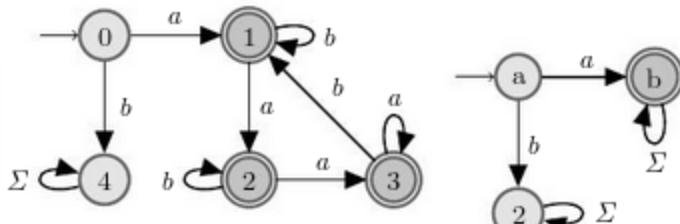
- Pour l'AFEND dans le tableau 7.1a, nous préférons utiliser la représentation tabulaire qui est plus simple à manipuler. Nous déterminisons l'automate en appliquant l'algorithme de déterminisation. Le résultat est représenté dans la figure 7.9a. Nous renommons les états dans la représentation tabulaire (figure 7.9b) et le représentons également sous forme graphique ci-dessous (figure 7.9c). Nous minimisons l'automate. Pour cet automate, l'opération de minimisation est évidente et nous nous passons d'appliquer l'algorithme de minimisation. Nous renommons les états de l'automate et représentons le résultat dans la figure 7.9d.
- Pour l'AFEND dans le tableau 7.1b, nous utilisons la représentation tabulaire. Nous déterminisons l'automate en appliquant l'algorithme de déterminisation. Le résultat est représenté dans la figure 7.9e. Nous renommons les états de l'automate et représentons le résultat dans la figure 7.9f.

	$\downarrow \{0\}$	$\{1, 2, 3, 4, 5\}^*$	$\{0, 1, 2, 3, 4\}^*$	$\{0, 1, 2, 3, 4, 5\}^*$	\emptyset
a	$\{1, 2, 3, 4, 5\}$	$\{0, 1, 2, 3, 4\}$	$\{0, 1, 2, 3, 4, 5\}$	$\{0, 1, 2, 3, 4, 5\}$	\emptyset
b	\emptyset	$\{1, 2, 3, 4, 5\}$	$\{1, 2, 3, 4, 5\}$	$\{1, 2, 3, 4, 5\}$	\emptyset

(a) AFED résultant de la déterminisation de l'AFED dans le tableau 7.1a.

	$\downarrow 0$	1^*	2^*	3^*	4
a	1	2	3	3	4
b	4	1	2	1	4

(b) AFED après renommage des états de l'AFED de la figure 7.9a - représentation tabulaire.



(c) AFED après renommage des états de l'AFED de la figure 7.9a - représentation graphique.

(d) Minimisé de l'AFED dans la figure 7.9c.

	$\downarrow \{0\}^*$	$\{1, 2\}$	$\{3, 4\}^*$	$\{5\}$	\emptyset
a	$\{1, 2\}$	\emptyset	$\{5\}$	\emptyset	\emptyset
b	\emptyset	$\{3, 4\}$	\emptyset	$\{0\}$	\emptyset

(e) AFED résultant de la déterminisation de l'AFED dans le tableau 7.1b.

	$\equiv 0$	$\equiv 1$	$\equiv 2$
2	2	2	2
0	0	0	0
1	1	1	1
3	3	3	3
4	4	4	4

(g) Minimisation de l'AFED dans la figure 7.9f.

	$\downarrow 0^*$	1	2^*	3	4
a	1	4	3	4	4
b	4	2	4	0	4

(f) AFED après renommage des états de l'AFED de la figure 7.9e - représentation tabulaire.

	$\downarrow 02^*$	13	4
a	1	4	4
b	4	2	4

(h) Minimisé de l'AFED dans la figure 7.9f.

Figure 7.9 – Résultats pour l'exercice 72 (p. 148).

Nous appliquons l'algorithme de minimisation dont l'exécution est représenté dans la figure 7.9g et représentons le minimisé sous forme tabulaire dans la figure 7.9h.

Solution de l'exercice 73 (page 149)

Pour les AFEND de cet exercice, nous utilisons la représentation tabulaire. Nous déterminons l'automate en appliquant l'algorithme de déterminisation.

1. Pour l'AFEND dans la figure 7.2a (resp. figure 7.2b, figure 7.2c), le déterminisé est représenté dans la figure 7.10a (resp. figure 7.10b, figure 7.10c).

	$\downarrow 1$	1, 2	1, 2, 3*	1, 3*
a	1, 2	1, 2, 3	1, 2, 3	1, 2
b	1	1, 3	1, 3	1

(a) Déterminisé de l'AFEND dans la figure 7.2a.

	$\downarrow 1*$	2, 3	3
a	\emptyset	1	\emptyset
b	2, 3	3	\emptyset

(b) Déterminisé de l'AFEND dans la figure 7.2b.

	$\downarrow 1$	1, 2	2, 3*
a	1, 2	1, 2	\emptyset
b	\emptyset	2, 3	2, 3

(c) Déterminisé de l'AFEND dans la figure 7.2c.

Figure 7.10 – Résultats pour l'exercice 73 (p. 149).

Solution de l'exercice 74 (page 149)

Pour les AFEND de cet exercice, nous utilisons la représentation tabulaire. Nous déterminons l'automate en appliquant l'algorithme de déterminisation.

1. Pour l'automate de la figure 7.3a (resp. figure 7.3b), le déterminisé est représenté dans la figure 7.11a (resp. figure 7.11b).

	$\downarrow 1*$	3	2	1, 3*	2, 3
a	3	3	1, 3	3	1, 3
b	2	3	3	2, 3	3

(a) Déterminisé de l'AFEND dans la figure 7.3a.

	$\downarrow 1*$	2, 3	3
a	\emptyset	1	\emptyset
b	2, 3	3	\emptyset

(b) Déterminisé de l'AFEND dans la figure 7.3b.

Figure 7.11 – Résultats pour l'exercice 74 (p. 149).

7.5.3 Déterminer l'équivalence entre deux AFEND

Solution de l'exercice 75 (page 149)

1. Les procédures pour déterminer l'équivalence définies pour les AFED prennent en entrée deux AFED et retournent un booléen. Les procédures à définir sur les AFEND prennent deux AFEND en entrée et retournent également un booléen. Nous obtenons de telles procédures simplement en réutilisant les procédures définies sur les AFED en leur passant les deux automates déterministes correspondant à la déterminisation des automates donnés en entrée.

Solution de l'exercice 76 (page 150)

1. Les deux automates sont déterministes et acceptent tous les mots qui contiennent une concaténation

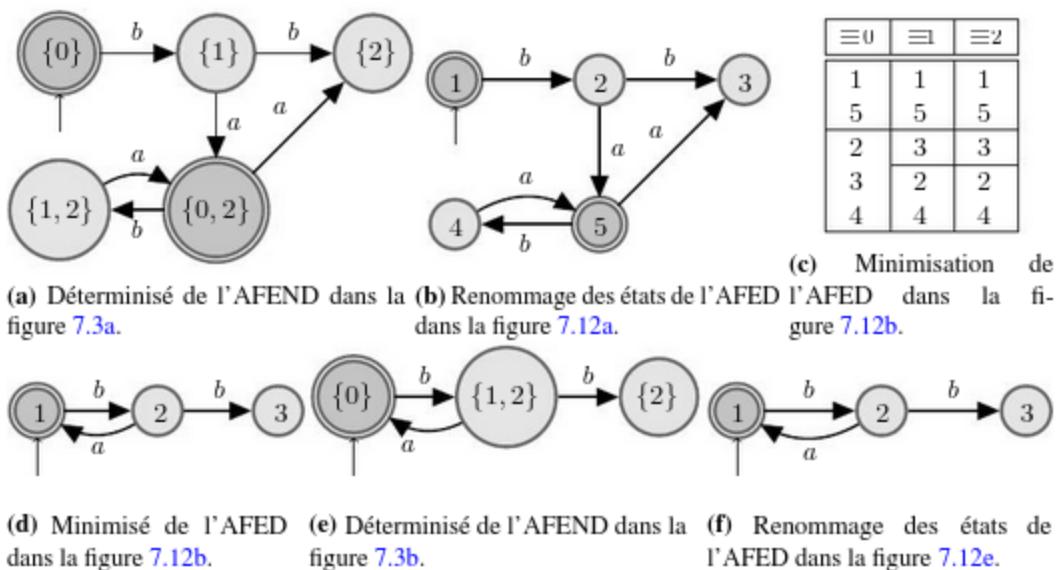


Figure 7.12 – Résultats pour l'exercice 76 (p. 150).

- Les deux automates reconnaissent le même langage : l'ensemble des mots qui contiennent une concaténation du mot/facteur $b \cdot a$.
- Nous choisissons de montrer l'équivalence entre les deux automates en utilisant la procédure basée sur la minimisation. Pour rappel, deux automates sont équivalents suivant cette procédure si et seulement si leur deux automates minimisés sont égaux modulo un renommage des états.
 - Déterminisons le premier automate en appliquant l'algorithme de déterminisation. Le résultat est représenté dans la figure 7.12a. Nous renommons les états de cet automate et représentons le résultat dans la figure 7.12b.
 - Nous appliquons l'algorithme de minimisation (voir figure 7.12c) et représentons l'automate minimisé dans la figure 7.12d.
 - Déterminisons le second automate en appliquant l'algorithme de déterminisation. Le résultat est représenté dans la figure 7.12e. Nous renommons les états de cet automate et représentons le résultat dans la figure 7.12f.
 - Nous remarquons immédiatement que les deux automates sont les mêmes. Formellement, nous aurions du tester l'équivalence entre les deux automates en utilisant le test d'équivalence vu dans le chapitre précédent.

7.5.4 Complexité

Solution de l'exercice 77 (page 150)

- Les exécutions correspondant au mot $abba$ et acceptées par l'automate sont données ci-dessous :

- $(1, abaa) \cdot (1, baa) \cdot (1, aa) \cdot (1, a) \cdot (2, \epsilon)$
- $(1, abaa) \cdot (1, baa) \cdot (1, aa) \cdot (2, a) \cdot (2, \epsilon)$
- $(1, abaa) \cdot (2, baa) \cdot (2, aa) \cdot (2, a) \cdot (2, \epsilon)$

Nous avons $\mathcal{N}(A_1, a \cdot b \cdot a \cdot a) = 3$.

2. Par induction sur $u \in \{a, b\}^*$:

- Cas de base : Nous avons $\mathcal{N}(A_1, \epsilon) = 0 = |\epsilon|_a$.
- Pas d'induction : Soit $u \in \{a, b\}^*$, supposons que $\mathcal{N}(A_1, u) = |u|_a$. Considérons un mot $u \cdot s \in \{a, b\}^*$ (on a $s \in \{a, b\}$). Nous distinguons deux cas selon que $s = a$ ou $s = b$:
- Cas $s = a$. Considérons l'ensemble des exécutions de u sur A_1 . Nous pouvons partitionner cet ensemble en distinguant les exécutions qui terminent dans l'état 1 et les exécutions qui terminent dans l'état 2. L'automate étant complet sur $\{a, b\}$ il n'y a pas d'autre exécution.

Considérons les exécutions qui terminent dans l'état 2. Chaque exécution de u qui termine dans l'état 2 (et qui est acceptée) peut être prolongée en une exécution qui termine dans l'état 2. Cette prolongation de l'exécution est faite de manière unique car il n'y a qu'une transition de l'état 2 sur lui-même avec le symbole a .

Considérons les exécutions qui terminent dans l'état 1. Comme il n'y a pas de transition depuis l'état 2 vers l'état 1, aucune de ces exécutions n'implique la transition de l'état 1 vers l'état 2. Ainsi, cet ensemble d'exécutions contient une unique exécution qui terminent dans l'état 1. Comme il y a deux transitions depuis l'état 1 sur le symbole a , cette exécution peut être prolongées de deux manières : en une exécution qui termine dans l'état 1 et une exécution qui termine dans l'état 2. L'exécution qui termine dans l'état 2 est acceptée. De plus, elle est différente de toutes les exécutions de $u \cdot a$ qui sont des prolongations d'exécutions de u qui terminent dans l'état 2 (car il n'y a pas de transitions depuis l'état 2 vers l'état 1). Ainsi, l'unique exécution acceptée de $u \cdot a$ construite à partir de l'exécution de u non acceptée s'ajoute aux exécutions acceptées de $u \cdot a$ qui sont des prolongations des exécutions de u acceptées. Nous avons donc $\mathcal{N}(A_1, u \cdot a) = \mathcal{N}(A_1, u) + 1$.

- Cas $s = b$. Comme le dernier symbole de $u \cdot b$ est b , les exécutions de $u \cdot b$ qui sont acceptées sont celles de u (qui terminent dans l'état 2) et qui sont prolongées en une exécution de $u \cdot b$. Ainsi, $\mathcal{N}(A_1, u) = \mathcal{N}(A_1, u \cdot b)$ et la propriété est vérifiée car $|u \cdot b|_a = |u|_a$.

Dans les deux cas, nous avons la propriété attendue.

3. Nous avons $\mathcal{N}(A_2, u) \leq 1$, pour tout mot u sur l'alphabet $\{a, b\}$, car l'automate est déterministe.
4. Pour chaque état et chaque symbole, nous avons deux transitions possibles depuis l'état. Ainsi, pour chaque exécution d'un mot u dans l'automate, cette exécution peut être prolongée en une exécution pour le mot $u \cdot s$ de deux manières possibles. Ainsi, le nombre d'exécutions de $u \cdot s$ est le double de celui pour u . Ainsi, nous avons $\mathcal{N}(A_3, u) = 2^{|u|}$.

5. En adaptant la construction pour le produit aux AFEND. Chaque exécution de l'automate produit est obtenue de manière unique à partir d'une exécution de A_1 et une exécution de A_2 .
6. Comme nous avons montré que $\mathcal{N}(A_1, u) = |u|_a$ pour tout mot $u \in \Sigma^*$, un automate répondant à la question est : $A_1 \times A_1$.

Solution de l'exercice 78 (page 150)

1. Les AFED qui reconnaissent L_1 , L_2 et L_3 sont représentés dans les figures 7.13a, 7.13b et 7.13c, respectivement. Le nombre d'états pour les AFED qui reconnaissent L_1 , L_2 et L_3 sont respectivement 2, 4 et 8.
2. Nous pouvons extrapoler que le nombre d'états d'un AFED reconnaissant L_n est 2^n .

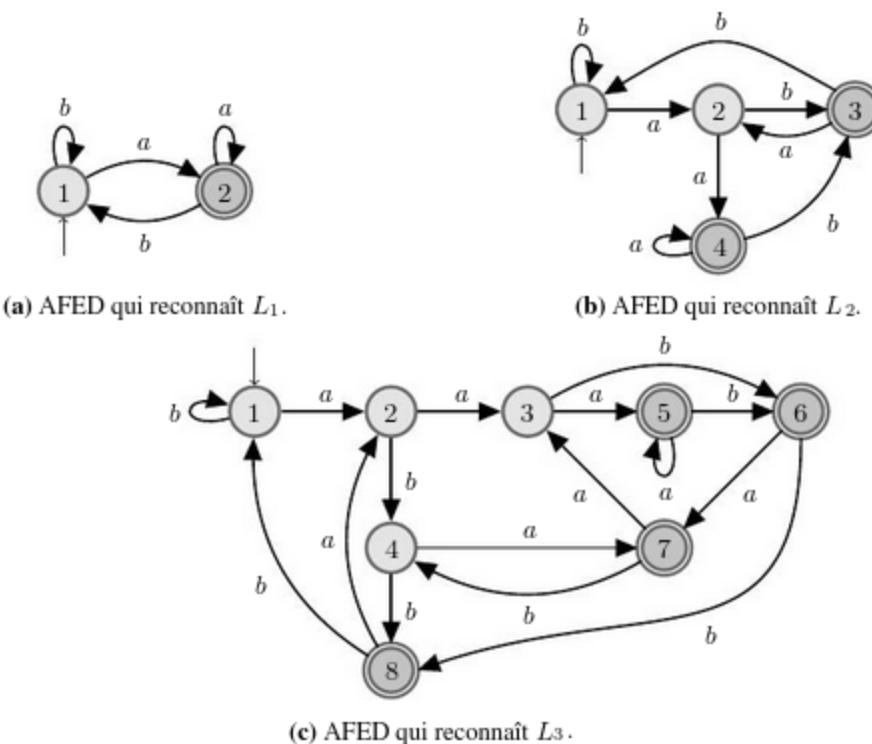


Figure 7.13 – Automates solutions de l'exercice 78.

Solution de l'exercice 79 (page 151)

1. Au pire des cas, tous les états du déterminisé sont accessibles. Si l'ensemble des états du non déterministe utilisé en paramètre est Q , alors l'ensemble des états du déterminisé est $\mathcal{P}(Q)$, l'ensemble des parties de Q . Si $|Q| = n$, alors $|\mathcal{P}(Q)| = 2^n$.
2. L'algorithme 32 réalise une déterminisation « à la volée » de l'AFEND passé en paramètre en réalisant une construction et un parcours des états du déterminisé « à la volée ». L'opération de visite consiste à découvrir de nouveaux états en utilisant la définition de la fonction de transition du déterminisé.

Algorithme 32 *déterminisation()* pour déterminiser « à la volée » un AFEND

Entrée : $A = (Q, \Sigma, q_{\text{init}}^N, \Delta, F)$ (* un AFEND *) .
Sortie : $A_d = (Q_d, \Sigma, q_{\text{init}}^D, \delta, F_d)$ (* un AFED équivalent au déterminisé de A ($\det(A)$) *)

```

1: ensemble d'états Accessibles,  $\mathbb{A}_{\text{visiter}}$ , Déjà_visités ;
2: état  $q_{\text{init}}^D := \{q_{\text{init}}^N\}$ ; (* état initial du déterminisé *)
3: ensemble d'états  $Q_d := \{q_{\text{init}}^D\}$  (* déclaration et initialisation de l'ensemble d'états de  $A_d$  *)
4: ensemble d'états  $F_d := \emptyset$  (* déclaration et initialisation de l'ensemble des états accepteurs de  $A_d$  *)
5:  $\mathbb{A}_{\text{visiter}} := \{q_{\text{init}}^D\}$ ; (* initialisation de  $\mathbb{A}_{\text{visiter}}$  *)
6:  $\text{Déjà\_visités} := \emptyset$ ; (* initialisation de Déjà_visités *)
7: tant que  $\mathbb{A}_{\text{visiter}} \neq \emptyset$  faire
8:   soit  $q_{\text{tmp}} \in \mathbb{A}_{\text{visiter}}$ ; (* on prend un état à visiter *)
9:    $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \setminus \{q_{\text{tmp}}\}$ ;
10:   $\text{Déjà\_visités} := \text{Déjà\_visités} \cup \{q_{\text{tmp}}\}$ ;
11:  si  $q_{\text{tmp}} \cap F \neq \emptyset$  alors (* est-ce que  $q_{\text{tmp}}$  contient un état accepteur de  $A$  ? *)
12:     $F_d := F_d \cup \{q_{\text{tmp}}\}$ ; (* ajout de  $q_{\text{tmp}}$  aux états accepteurs de  $A_d$  *)
13:  fin si
14:  pour  $s \in \Sigma$  faire (* utilisation de la définition de la fonction de transition du déterminisé *)
15:     $q_{\text{succ}} := \{q' \in Q \mid \exists q \in q_{\text{tmp}} . (q, s, q') \in \Delta\}$ ; (* successeur de  $q_{\text{tmp}}$  sur  $s$  *)
16:     $Q_d := Q_d \cup \{q_{\text{succ}}\}$ ; (* ajout de  $q_{\text{succ}}$  aux états de  $A_d$  *)
17:     $\delta := \delta \cup \{(q_{\text{tmp}}, s, q_{\text{succ}})\}$ ; (* ajout de la transition depuis  $q_{\text{tmp}}$  vers  $q_{\text{succ}}$  sur  $s$  *)
18:    si  $q_{\text{succ}} \notin \text{Déjà\_visités}$  alors
19:       $\mathbb{A}_{\text{visiter}} := \mathbb{A}_{\text{visiter}} \cup \{q_{\text{succ}}\}$ ; (*  $q_{\text{succ}}$  est à visiter *)
20:    fin si
21:  fin pour
22: fin tant que
23: retourner  $(Q_d, \Sigma, q_{\text{init}}^D, \delta, F_d)$ ;
```

3. Le pire des cas se produit lorsque tous les états de l'automate sont marqués comme à visiter. C'est-à-dire lorsque à chaque itération, il y a au moins un nouvel état qui n'a pas été déjà visité qui est marqué comme à visiter. Cela peut être fait au plus $2^{|Q_N|}$ fois ; c'est le cardinal de l'ensemble des sous ensembles de Q_N .
4. En suivant le raisonnement de la question précédente, le corps de la boucle s'exécute $2^{|Q_N|}$ fois au maximum. Les trois premières instructions s'exécuteront $2^{|Q_N|}$ fois. Les instructions dans le corps de la boucle s'exécuteront $|\Sigma|$ fois par itération.
5. Nous donnons dans le tableau ci-dessous quelques valeurs de la fonction $\mathbb{N} \rightarrow \mathbb{N}$ définie par $x \mapsto 2^x$ en certains points.

x	5	10	20	30
2^x	32	$\approx 10^3$	$\approx 10^6$	$\approx 10^9$

7.5.5 Algorithmes pour les AFEND

Solution de l'exercice 80 (page 151)

1. L'algorithme 33 prend en entrée un AFEND défini sur un alphabet Σ , un alphabet Σ' et renvoie un booléen indiquant si l'AFEND est un AFED sur l'alphabet Σ' .

Notons que la réponse à la question et l'algorithme ne dépendent pas de la relation (ensembliste) entre Σ et Σ' . L'algorithme parcourt les états de l'automate. Pour chaque état q , pour chaque symbole s de l'alphabet Σ' , on calcule Out_q^s , l'ensemble des transitions sortant de l'état q sur le symbole s . Si le cardinal de cet ensemble est strictement supérieur à 1, l'algorithme termine en retournant faux. Si l'itération sur les états termine (sans retourner faux), alors il n'y a aucun état et aucun symbole tel qu'il y ait (au moins) deux transitions sortantes sur le même symbole. L'algorithme retourne vrai et l'automate est déterministe sur Σ' .

Algorithme 33 *est_déterministe_alpha*(A) pour déterminer si un AFEND est un AFED sur un alphabet donné

Entrée : $A = (Q, \Sigma, q_{\text{init}}^N, \Delta, F)$ (* un AFEND sur un alphabet Σ *).
Entrée : Σ' (* un alphabet *).
Sortie : vrai si A est un AFED sur l'alphabet Σ' , faux sinon.

```

pour  $q \in Q$  faire (* pour chaque état  $q$  de l'automate *)
    pour  $s \in \Sigma'$  faire (* pour chaque symbole  $s$  de l'alphabet d'entrée  $\Sigma'$  *)
         $Out_q^s := (\{q\} \times \{s\} \times Q) \cap \Delta$ ; (*  $Out_q^s$  est l'ensemble des transitions à partir de  $q$  sur  $s$ . *)
        si  $|Out_q^s| > 1$  alors
            retourner faux;
            (*  $A$  n'est pas déterministe car il y a plusieurs transitions sur  $s$  à partir de  $q$  *)
        fin si
    fin pour
fin pour
retourner vrai;
```

2. Nous réutilisons directement l'algorithme donné à la réponse précédente avec comme alphabet paramètre, l'alphabet de l'automate. Nous obtenons l'algorithme 34.

Algorithme 34 *est_déterministe()* pour déterminer si un AFEND est un AFED sur un alphabet donné

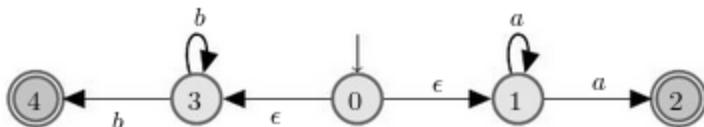
Entrée : $A = (Q, \Sigma, q_{\text{init}}^N, \Delta, F)$ (* un AFEND sur un alphabet Σ *).
Sortie : vrai si A est un AFED, faux sinon.
retourner *est_deterministe*(A, Σ);

Automates non déterministes avec ϵ -transitions

8.1 Résumé intuitif du chapitre

Dans ce chapitre, nous nous intéressons à la notion d'**automates non déterministes avec ϵ -transitions**. Nous introduisons des transitions étiquetées par le mot de longueur 0 (noté ϵ pour rappel). Pour une transition $p \xrightarrow{\epsilon} q$, cela permet de passer de l'état p à l'état q spontanément. Du point de vue de la machine, où les éléments du vocabulaire représentent une action de l'environnement, cela l'autorise à évoluer de l'état p à l'état q « silencieusement ».

Par exemple, dans l'exemple suivant, l'automate reconnaît les mots contenant soit un nombre quelconque de a ou soit un nombre quelconque de b ; ces nombres devant être strictement positifs.



Comme pour les automates déterministes et non déterministes, on définit les notions de **configurations**, d'**exécution**, de **mot** et de **langage acceptés** (ou reconnu). Finalement, l'introduction d' ϵ -transitions n'augmente pas l'expressivité en termes de langage reconnu : nous introduisons la notion **ϵ -fermeture** et d'**élimination des ϵ -transitions** qui permettent de passer d'un automate non déterministe avec ϵ -transitions à un automate non déterministe qui reconnaît le même langage. Cette opération peut être suivie de **déterminisation**.

Finalement, la **classe des langages à états est fermée** par union, intersection, complémentation, concaténation, image miroir, morphisme et morphisme inverse ainsi que par fermeture de Kleene.

8.2 Les notions essentielles

Dans la suite, nous considérons un alphabet Σ qui ne contient pas le symbole ϵ ($\epsilon \notin \Sigma$).

8.2.1 Définition et langage reconnu

Définition 114 (Automate non déterministe avec ϵ -transitions, ϵ -AFEND) *Un automate à nombre fini d'états et non déterministe avec ϵ -transitions, abrégé ϵ -AFEND, est un quintuplet $(Q, \Sigma, q_{\text{init}}, \Delta, F)$ où :*

- Q est un ensemble fini d'états,
- Σ est l'alphabet de l'automate,
- $q_{\text{init}} \in Q$ est l'état initial,
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ est la relation de transition,
- $F \subseteq Q$ est l'ensemble des états accepteurs.

Dans la suite, nous considérons un ϵ -AFEND $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$.

Définition 115 (Configuration d'un ϵ -AFEND) *Une configuration de l'automate A est un couple (q, u) où $q \in Q$ et $u \in \Sigma^*$.*

Définition 116 (Relation de dérivation d'un ϵ -AFEND) *La relation de dérivation entre configurations, notée \rightarrow_Δ , est l'ensemble :*

$$\{((q, s \cdot u), (q', u')) \mid u \in \Sigma^* \wedge (((q, s, q') \in \Delta \wedge u' = u) \vee (s \cdot u = u' \wedge (q, \epsilon, q') \in \Delta))\}.$$

Lorsque $((q, s \cdot u), (q', u')) \in \rightarrow_\Delta$, on note $(q, s \cdot u) \rightarrow_\Delta (q', u')$.

On définit la notation $q \xrightarrow[\Delta]{u} q'$ inductivement comme suit : pour tout état $q, q' \in Q$, mot $u \in \Sigma^*$ et symbole $s \in \Sigma$,

- $q \xrightarrow[\Delta]{\epsilon} q$, et
- $q \xrightarrow[\Delta]{u \cdot s} q'$ si il existe $q'' \in Q$ tel que $q \xrightarrow[\Delta]{u} q''$ et $(q'', s, q') \in \Delta$.

Les notions d'exécution, d'acceptation d'un mot et de langage reconnu sont définies comme dans le cas des AFEND mutatis mutandis.

8.2.2 Élimination des ϵ -transitions

Traduction vers AFEND

Définition 117 (Élimination des ϵ -transitions) *L'AFEND associé à A par élimination des ϵ -transitions, noté $\epsilon\ell(A)$, est $\epsilon\ell(A) = (Q, \Sigma, q_{\text{init}}, \epsilon\ell(\Delta), \epsilon\ell(F))$ avec :*

- la relation de transition définie par :

$$\epsilon\ell(\Delta) = \left\{ (q, s, q') \mid \exists q_1, q_2 \in Q. \left(q \xrightarrow[\Delta]{\epsilon} q_1 \wedge (q_1, s, q_2) \in \Delta \wedge q_2 \xrightarrow[\Delta]{\epsilon} q' \right) \right\}$$

- l'ensemble des états accepteurs défini par :

$$\epsilon\ell(F) = \left\{ q \in Q \mid \exists q' \in F. q \xrightarrow{\epsilon}^* q' \right\}.$$

Proposition 4 (Correction de la procédure d'élimination des ϵ -transitions)

$$\mathcal{L}_{\text{auto}}(A) = \mathcal{L}_{\text{auto}}(\epsilon\ell(A)).$$

Traduction (directe) vers AFED

Définition 118 (ϵ -Fermeture d'un état) Soit $q \in Q$ un état, l' ϵ -fermeture de q est l'ensemble d'états défini inductivement comme suit :

- Cas de base : $\{q\}$,
- Pas d'induction : $p \mapsto \{r \in Q \mid (p, \epsilon, r) \in \Delta\}$.

On note $\text{e-fermeture}(q)$ l' ϵ -fermeture de l'état q .

Le cas de base indique que $q \in \text{e-fermeture}(q)$. Le pas d'induction indique que si un état $p \in \text{e-fermeture}(q)$ et s'il existe une transition de p vers un état $r \in Q$ étiquetée par ϵ , alors $r \in \text{e-fermeture}(q)$.

Remarque 33 De manière équivalente, $\text{e-fermeture}(q) = \delta^*(q, \epsilon)$ et $\text{e-fermeture}(q) = \left\{ q' \mid q \xrightarrow{\epsilon}^* q' \right\}$.

Définition 119 (ϵ -Fermeture d'un ensemble d'états) L' ϵ -fermeture d'un ensemble d'états $S \subseteq Q$, notée $\text{e-fermeture}(S)$, est l'ensemble d'états défini comme suit :

$$\text{e-fermeture}(S) = \bigcup_{q \in S} \text{e-fermeture}(q).$$

Définition 120 (Déterminisation et élimination des ϵ -transitions, « à la volée ») Le déterminisé de A est l'ADEF, noté $\det(A)$, et défini par $(Q^D, \Sigma, q_{\text{init}}^D, \delta^D, F^D)$ avec :

- $Q^D = \mathcal{P}(Q)$,
- $q_{\text{init}}^D = \text{e-fermeture}(q_{\text{init}})$,
- δ^D est définie comme suit : pour tout $S \in Q^D, s \in \Sigma$:
 - soit $\{p_1, p_2, \dots, p_k\} = S$,
 - soit $\{r_1, r_2, \dots, r_m\} = \bigcup_{i=1}^k \{p'_i \mid (p_i, s, p'_i) \in \Delta\}$,
 - alors $\delta(S, s) = \text{e-fermeture}(\{r_1, r_2, \dots, r_m\})$,
- $F^D = \{S \in \mathcal{P}(Q) \mid S \cap F \neq \emptyset\}$.

Remarque 34 Chaque état de l'automate déterminisé (atteint avec δ^D) correspond à un ensemble d'états de l'automate non déterministe avec ϵ -transitions qui est ϵ -fermé : $\forall S \in Q^D. \text{e-fermeture}(S) = S$.

Proposition 5 (Correction de la procédure de déterminisation)

$$\mathcal{L}_{\text{auto}}(A) = \mathcal{L}_{\text{auto}}(\det(A)).$$

Remarque 35 En conséquence des propriétés 4 et 5, nous avons deux méthodes pour passer d'un ϵ -AFEND à un AFED :

1. transformer l' ϵ -AFEND en AFEND en supprimant les ϵ -transitions (définition 117), puis transformer l'automate obtenu en AFED par déterminisation (définition 112) ;
2. transformer l' ϵ -AFEND en AFED en utilisant la déterminisation et élimination ϵ -transitions à la volée (définition 120).

8.2.3 Retour sur la fermeture de la classe des langages à états

Dans le chapitre 4, nous avons la fermeture de la classe des langages à états par complémentation et intersection. Dans la suite, nous considérons des langages L , L_1 et L_2 sur des alphabets Σ , Σ_1 , et Σ_2 respectivement.

Fermeture par union et concaténation

Proposition 6 (Fermeture des langages à états par union et concaténation) Les langages à états sont fermés par les opérateurs d'union ensembliste et de concaténation de langages. C'est-à-dire, si L_1 et L_2 sont des langages à états, alors $L_1 \cup L_2$ et $L_1 \cdot L_2$ sont des langages à états :

$$\begin{aligned} \forall L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^* . L_1 \in EF(\Sigma_1) \wedge L_2 \in EF(\Sigma_2) &\implies L_1 \cup L_2 \in EF(\Sigma_1 \cup \Sigma_2) \\ \forall L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^* . L_1 \in EF(\Sigma_1) \wedge L_2 \in EF(\Sigma_2) &\implies L_1 \cdot L_2 \in EF(\Sigma_1 \cup \Sigma_2) \end{aligned}$$

La première proposition exprime la fermeture par union, la seconde exprime la fermeture par concaténation.

Démontrer ces propositions fait l'objet des exercices 97 (p. 177) et 98 (p. 177).

Fermeture par opération miroir

Définition 121 (Mot miroir) Le mot miroir du mot $u = a_1 \cdot a_2 \cdots a_n$, noté u^R , est le mot défini par $a_n \cdot a_{n-1} \cdots a_1$.

Le mot miroir (ou image miroir d'un mot) est le mot que l'on obtient en lisant le mot u de droite à gauche (comme en arabe ou en hébreu). Dans l'exercice 102, nous verrons une définition équivalente qui utilise la définition inductive de mot.

Définition 122 (Langage miroir) Le langage miroir du langage $L \subseteq \Sigma^*$ (ou image de L par opération miroir), noté L^R , est le langage défini par $L^R = \{w^R \mid w \in L\}$.

Proposition 7 (Fermeture des langages à états par l'opération miroir) Les langages à états sont fermés par l'opération miroir. C'est-à-dire, si L est un langage à états, alors L^R est un langage à états :

$$\forall L \subseteq \Sigma^* . L \in EF(\Sigma) \implies L^R \in EF(\Sigma).$$

Démontrer cette proposition fait l'objet de l'exercice 102 (p. 178).

Fermeture par morphisme

Pour la fermeture par morphisme, nous utilisons la notion de groupe, définition 36 (p. 25). Dans la suite, pour chaque alphabet Σ , nous considérons le groupe $(\Sigma^*, \cdot, \epsilon)$ où \cdot est l'opération de concaténation entre mots de Σ^* et ϵ le mot vide. Nous utilisons des morphismes pour traduire des mots sur un alphabet vers des mots sur un autre alphabet. Par ailleurs, nous considérons deux alphabets Σ et Σ' .

Définition 123 (Morphisme de mots) Une application $h : \Sigma \rightarrow \Sigma'^*$ induit un morphisme de mots $\hat{h} : \Sigma^* \rightarrow \Sigma'^*$ défini par $\hat{h}(\epsilon) = \epsilon$ et $\hat{h}(u \cdot a) = \hat{h}(u) \cdot h(a)$.

Remarque 36 On peut montrer que \hat{h} est effectivement un morphisme en démontrant $\forall x, y \in \Sigma^*. \hat{h}(x \cdot y) = \hat{h}(x) \cdot \hat{h}(y)$ par induction sur y .

Pour simplifier les notations, nous confondons l'application induisant un morphisme et le morphisme induit, et nous écrivons h au lieu de \hat{h} .

Dans la suite, nous considérons un morphisme h .

Définition 124 (Image d'un langage par un morphisme) L'image de L par un morphisme h , notée $h(L)$, est définie par $\{h(u) \mid u \in L\}$.

Proposition 8 (Fermeture des langages à états par morphisme) Les langages à états sont fermés par morphisme. C'est-à-dire, si L est un langage à états, alors $h(L)$, son image par h , est un langage à états :

$$\forall L \subseteq \Sigma^*. L \in EF(\Sigma) \implies h(L) \in EF(\Sigma).$$

Démontrer cette proposition fait l'objet de l'exercice 103 (p. 178).

Fermeture par morphisme inverse

Soit h^{-1} le morphisme inverse du morphisme h (application inverse).

Définition 125 (Image d'un langage par un morphisme inverse) L'image d'un langage L par l'inverse h^{-1} d'un morphisme h , notée $h^{-1}(L)$, est définie par $\{u \in \Sigma^* \mid \exists u' \in L. h(u) = u'\}$.

Proposition 9 (Fermeture des langages à états par morphisme inverse) Les langages à états sont fermés par morphisme inverse. C'est-à-dire, si L est un langage à états, alors $h^{-1}(L)$, son image par h^{-1} , est un langage à états :

$$\forall L \subseteq \Sigma^*. L \in EF(\Sigma) \implies h^{-1}(L) \in EF(\Sigma).$$

Démontrer cette proposition fait l'objet de l'exercice 104 (p. 178).

Fermeture de Kleene

Définition 126 (Fermeture de Kleene) La fermeture de Kleene de L , notée L^* , est l'ensemble défini inductivement comme suit :

- Cas de base : ϵ ,

- Pas d'induction : application de $L \times L^* \rightarrow L^*$ définie par $u, v \mapsto u \cdot v$.

Remarque 37 (Définitions équivalentes de la fermeture de Kleene) Nous avons les deux définitions équivalentes suivantes de la fermeture de Kleene :

- La fermeture de Kleene de L est le plus petit ensemble généré par les deux règles suivantes :
 - $\epsilon \in L^*$, et
 - si $u \in L$ et $v \in L^*$, alors $u \cdot v \in L^*$ (de manière équivalente : si $u \in L^*$ et $v \in L$, alors $u \cdot v \in L^*$).
- La fermeture de Kleene de L est l'ensemble des mots formés par un nombre fini de concaténations de mots de L :

$$L^* = \{\epsilon\} \cup \{w_0 \cdots w_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n \implies w_i \in L\}.$$

Proposition 10 (Fermeture des langages à états par la fermeture de Kleene) Les langages à états sont fermés par la fermeture de Kleene. C'est-à-dire, si L est un langage à états, alors L^* est un langage à états :

$$\forall L \subseteq \Sigma^*. L \in EF(\Sigma) \implies L^* \in EF(\Sigma).$$

Résumé des fermetures

La tableau 8.1 résume les propriétés de fermeture des langages à états.

8.3 Exercices

Exercice 81 (♣) — Mot accepté et mot non accepté par un ϵ -AFEND

Nous considérons l'alphabet $\{a, b\}$ et l' ϵ -AFEND dans la figure 8.1b.

1. Donner un mot accepté par cet AFEND et représenter son exécution.
2. Donner un mot non accepté par cet AFEND et représenter son exécution.

Exercice 82 (♣♣♣) — Fonction de transition étendue pour un ϵ -AFEND

Nous considérons un ϵ -AFEND $(Q, \Sigma, q_{\text{init}}, \Delta, F)$.

1. À partir de la relation Δ , définir la fonction $\Delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$ qui à un état q et un mot w associe l'ensemble des états accessibles en lisant w à partir de q avec Δ .
2. Étendre la fonction trouvée à la question précédente pour définir la fonction $\Delta^* : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$ qui à un ensemble d'états Q et un mot w associe l'ensemble des états accessibles en lisant w à partir de l'un des états de Q avec Δ .

Tableau 8.1 – Résumé des opérations de fermeture sur les langages à états avec leur opérations associées sur les automates. Dans le tableau, L , L_1 et L_2 désignent des langages et A , A_1 , A_2 désignent des automates à finis. Nous indiquons également les définitions et les exercices faisant référence à ces opérations.

Fermeture	Opération sur les langages	Opération sur les automates
Complémentation	\overline{L} définition 17 (p. 22)	\overline{A} définition 79 (p. 72)
Intersection	$L_1 \cap L_2$ définition 19 (p. 22)	$A_1 \times \cap A_2$ définition 80 (p. 73)
Union	$L_1 \cup L_2$ définition 18 (p. 22)	$A_1 \times \cup A_2$ exercice 97 (p. 177)
Concaténation	$L_1 \cdot L_2$ définition 59 (p. 31)	$A_1 \cdot A_2$ exercice 97 (p. 177)
Miroir	L^R définition 122 (p. 170)	A^R exercice 102
Morphisme	$h(L)$ définition 124 (p. 171)	$h(A)$ exercice 103 (p. 178)
Morphisme inverse	$h^{-1}(L)$ définition 125 (p. 171)	$h^{-1}(A)$ exercice 104 (p. 178)
Fermeture de Kleene	L^* définition 126 (p. 171)	A_* exercice 97 (p. 177)

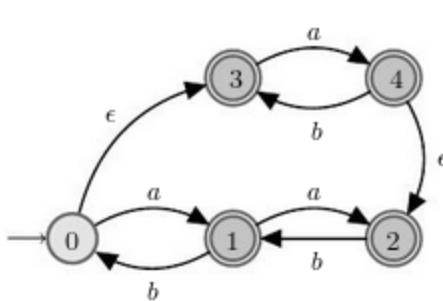
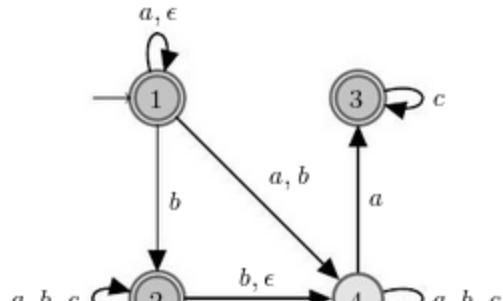
(a) ϵ -AFEND pour l'exercice 94.(b) ϵ -AFEND pour les exercices 81, 94 et 95.

Figure 8.1 – Des automates non déterministes avec ϵ -transitions en représentation graphique.

8.3.1 Trouver un automate

Exercice 83 (♣) — Reconnaître un mot étant donné deux états

Soit Σ un alphabet et w un mot défini sur l'alphabet Σ . Soient q et q' deux états.

- Définir un ϵ -AFEND $A_{q,w,q'}$ qui reconnaît le langage $\{w\}$ (contenant uniquement le mot w) tel que q soit l'état initial et q' l'unique état accepteur.

Exercice 84 (♠♠) — Nombres décimaux

Nous souhaitons obtenir un automate reconnaissant les nombres écrit en notation décimale. Nous considérons l'alphabet $\Sigma = \{0, \dots, 9, \bullet, +, -\}$ contenant les chiffres utilisés dans la représentation en base 10 des nombres décimaux, le symbole \bullet marquant le début la partie décimale, les symboles $+$ et $-$ pour indiquer le signe du nombre. Un nombre écrit en notation décimale est constitué des éléments suivants :

- un signe $+$ ou $-$ optionnel,
- un mot de numéros $0, 1, \dots, 9$ correspondant à la partie entière,
- un point (\bullet) pour marquer la décimale,
- un mot de numéros $0, 1, \dots, 9$ correspondant à la partie décimale.

De plus, l'un des deux mots de numéros peut être vide, mais ils ne peuvent pas être tous les deux vides.

1. Donner un ϵ -AFEND reconnaissant les nombres en notation décimales et suivant cette convention.
2. Remarquons que cette convention permet d'écrire des nombres comme $00 \bullet 01$ ou 02 . Pour remédier à cela, reprendre l'automate trouvé à la question précédente en distinguant l'entier commençant par 0 des autres entiers et les réels qui commencent par 0 mais avec un seul 0 avant le \bullet .

Exercice 85 (♠♠♣) — Entiers en base 2 et divisibilité par 3

Nous considérons l'alphabet $\Sigma = \{0, 1\}$ des entiers naturels représentés en binaire (c'est-à-dire en base 2). Les entiers dans cet exercice sont représentés en binaire.

1. En utilisant l'AFED reconnaissant les entiers divisibles par 3 avec la convention *gros-boutiste* trouvé dans l'exercice 22, donner un automate qui accepte les entiers divisibles par 3 et représentés avec la convention *petit-boutiste*, c'est-à-dire avec la lecture des bits de poids les plus faibles en premier.

Exercice 86 (♠♠♣) — Fermeture par suffixe et facteur

Soit L un langage sur un alphabet Σ . Nous considérons les langages $\text{Suf}(L)$ et $\text{Fact}(L)$ des fermetures par suffixe et facteur de L comme introduits dans la définition 60 (p. 31). Supposons que L soit un langage à états.

1. Démontrer que $\text{Suf}(L)$ est également un langage à états. Nous en déduisons que la classe des langages à états est fermée par (l'opération de fermeture par) suffixe.
2. Démontrer que $\text{Fact}(L)$ est également un langage à états. Nous en déduisons que la classe des langages à états est fermée par (l'opération de fermeture par) facteur.

Exercice 87 (♠♠♣) — Langage des moitiés de mots

Soit A un AFEND reconnaissant un langage L . Nous considérons le langage des « moitiés de mots de L », noté $L^{1/2}$, et défini comme suit :

$$L^{1/2} = \{u \in \Sigma^* \mid \exists v \in \Sigma^*. |u| = |v| \wedge u \cdot v \in L\}.$$

$L^{1/2}$ contient les mots u qui peuvent être étendus par un mot de même longueur v pour former un mot de L en concaténant v à u .

1. À partir de A qui reconnaît L , donner un ϵ -AFEND qui reconnaît $L^{1/2}$.
2. Démontrer que le langage reconnu par l' ϵ -AFEND obtenu à la question précédente est bien $L^{1/2}$.

8.3.2 Fermeture de Kleene

Exercice 88 (♠♦♣) — Concaténation et fermeture de Kleene

1. Démontrer que : $\forall L \subseteq \Sigma^*. L^* \cdot L^* = L^*$.

Exercice 89 (♠♦♣) — Idempotence de la fermeture de Kleene

En utilisant le résultat de l'exercice 88, nous souhaitons démontrer que la fermeture de Kleene est idempotente.

1. Démontrer que : $\forall L \subseteq \Sigma^*. L^* = (L^*)^*$.

Exercice 90 (♠♦♣) — Union et inclusion avec la fermeture de Kleene

Soient L_1 et L_2 des langages.

1. Démontrer que si $L_1 \subseteq L_2$, alors $L_1^* \subseteq L_2^*$.
2. En déduire que $L_1^* \cup L_2^* \subseteq (L_1 \cup L_2)^*$.
3. Montrer que, en général, $L_1^* \cup L_2^* \neq (L_1 \cup L_2)^*$.
4. Trouver des langages L_1 et L_2 tels que $L_1 \not\subseteq L_2$, $L_2 \not\subseteq L_1$ et $L_1^* \cup L_2^* = (L_1 \cup L_2)^*$.

Exercice 91 (♠♦♣) — Union et concaténation avec la fermeture de Kleene

1. Démontrer que : $\forall L_1, L_2 \subseteq \Sigma^*. (L_1 \cup L_2)^* = (L_1^* \cdot L_2^*)^*$ (en utilisant possiblement les résultats de l'exercice 88 et l'exercice 90).
2. Est-ce que : $\forall L_1, L_2 \subseteq \Sigma^*. (L_1 \cup L_2)^* = (L_1^* \cdot L_2)^*$? Si oui, faire une preuve, sinon, donner un contre-exemple.
3. Déterminer une condition nécessaire et suffisante pour que $(L_1 \cup L_2)^* = (L_1^* \cdot L_2)^*$.

8.3.3 Élimination des ϵ -transitions et déterminisation

Dans les exercices de cette section, nous utilisons :

- l'élimination des ϵ -transitions (définition 117), c'est-à-dire la transformation qui permet, à partir d'un ϵ -AFEND, d'obtenir un AFEND équivalent;
- la déterminisation (définition 112), c'est-à-dire la transformation qui permet, à partir d'un AFEND, d'obtenir un AFED équivalent;
- la déterminisation directe (définition 120), c'est-à-dire la transformation qui permet, à partir d'un ϵ -AFEND, d'obtenir un AFED équivalent.

Exercice 92 (♣♣) — Élimination des ϵ -transitions et déterminisation, utilisation de la représentation tabulaire

Considérons l'alphabet $\Sigma = \{a, b\}$ et l' ϵ -AFEND défini par $(\{0, 1, 2, 3, 4\}, \{a, b\}, 0, \Delta, \{4\})$ dont la relation de transition Δ est définie par le tableau 8.2a. Pour les questions suivantes, utiliser la représentation tabulaire des automates.

1. Éliminer les ϵ -transitions.
2. Déterminiser l'automate obtenu à la question précédente.
3. Déterminiser l'automate initial en utilisant la méthode directe (combinaison de l'élimination des ϵ -transitions et déterminisation).
4. Vérifier que les automates obtenus aux deux questions précédentes (c'est-à-dire en utilisant les deux méthodes) sont équivalents.

Tableau 8.2 – Des automates non déterministes avec ϵ -transitions en représentation tabulaire. Les états sont en colonnes, les symboles en ligne. L'étoile marque un état accepteur. L'état initial est l'état 0.

	$\downarrow 0$	1	2	3	4^*
ϵ	1, 3		4		
a		2		4	
b			1		3

 (a) ϵ -AFEND pour l'exercice 92.

	$\downarrow 0$	1	2	3	4^*
ϵ	1, 3	3	1		3
a		2			
b				4	

 (b) ϵ -AFEND pour l'exercice 93.

Exercice 93 (♣♣) — Élimination des ϵ -transitions et déterminisation, utilisation de la représentation tabulaire

Soit l' ϵ -AFEND défini par $(\{0, 1, 2, 3, 4\}, \{a, b\}, 0, \Delta, \{4\})$ dont la relation de transition Δ est définie par le tableau 8.2b. Pour les questions suivantes, utiliser la représentation tabulaire des automates.

1. Éliminer les ϵ -transitions.
2. Déterminiser l'automate obtenu à la question précédente.
3. Déterminiser l'automate initial en utilisant la méthode directe (combinaison de l'élimination des ϵ -transitions et déterminisation).
4. Vérifier que les automates obtenus aux deux questions précédentes (c'est-à-dire en utilisant les deux méthodes) sont équivalents.

Exercice 94 (♣♣) — Élimination des ϵ -transitions et déterminisation, utilisation de la représentation graphique

Considérons l'alphabet $\Sigma = \{a, b\}$ et l' ϵ -AFEND représenté dans la figure 8.1a, défini sur Σ . Pour les questions suivantes, utiliser la représentation graphique des automates.

1. Donner un mot accepté et un mot non accepté par l'automate.

2. Éliminer les ϵ -transitions.
3. Déterminiser l'automate obtenu à la question précédente.
4. Minimiser l'automate obtenu à la question précédente.

Exercice 95 (♠♠) — Élimination des ϵ -transitions et déterminisation

Soit $\Sigma = \{a, b, c\}$ et l' ϵ -AFEND dans la figure 8.1b défini sur Σ :

1. Éliminer les ϵ -transitions.
2. Déterminiser l'automate obtenu à la question précédente.
3. Minimiser l'automate obtenu à la question précédente.

8.3.4 Composition et transformation d' ϵ -AFEND

Exercice 96 (♠♠♠) — Montrer qu'un langage est un langage à états

Soit L un langage à états sur un alphabet Σ .

1. Démontrer que $\{w \mid s \cdot w \in L\}$, l'ensemble des mots de L commençant par un symbole $s \in \Sigma$ et le supprimant, est un langage à états.
2. Démontrer que $\{w \mid w \cdot s \in L\}$, l'ensemble des mots non vides de L terminant par un symbole $s \in \Sigma$ et le supprimant, est un langage à états.
3. Démontrer que $\{w \cdot s \cdot s \mid w \cdot s \in L\}$, l'ensemble des mots obtenus en doublant la dernière lettre des mots non vides de L , est un langage à états.
4. Démontrer que $\{w_1 \cdot s \cdot w_2 \mid w_1, w_2 \in L\}$, l'ensemble des mots obtenu en insérant une occurrence d'un symbole $s \in \Sigma$ dans un mot de L , est un langage à états.

Exercice 97 (♠♠♠) — Composition d' ϵ -AFEND

Soient $A_1 = (Q^1, \Sigma, q_{\text{init}}^1, \Delta^1, F^1)$ et $A_2 = (Q^2, \Sigma, q_{\text{init}}^2, \Delta^2, F^2)$ deux ϵ -AFEND et L_1, L_2 les langages reconnus par A_1, A_2 , respectivement.

1. Définir l'automate $A_{\cup} = (Q^{\cup}, \Sigma, q_{\text{init}}^{\cup}, \Delta^{\cup}, F^{\cup})$ qui reconnaît $L_1 \cup L_2$.
2. Définir l'automate $A_{\cdot} = (Q^{\cdot}, \Sigma, q_{\text{init}}, \Delta^{\cdot}, F^{\cdot})$ qui reconnaît $L_1 \cdot L_2$.
3. Définir l'automate $A_{*} = (Q^{*}, \Sigma, q_{\text{init}}^{*}, \Delta^{*}, F^{*})$ qui reconnaît L_1^* .
4. Définir l'automate $A_{\cap} = (Q^{\cap}, \Sigma, q_{\text{init}}^{\cap}, \Delta^{\cap}, F^{\cap})$ qui reconnaît $L_1 \cap L_2$.

Exercice 98 (♠♠♠) — Composition d' ϵ -AFEND - preuve de correction

Soient $A_1 = (Q^1, \Sigma, q_{\text{init}}^1, \Delta^1, F^1)$ et $A_2 = (Q^2, \Sigma, q_{\text{init}}^2, \Delta^2, F^2)$ deux ϵ -AFEND et L_1, L_2 les langages reconnus par A_1 et A_2 respectivement. Nous supposons les ensemble d'états Q^1 et Q^2 de ces automates disjoints, et ainsi évitons un éventuel problème de renommage des états. Nous considérons les compositions d'automates obtenues à l'exercice 97. Nous voulons démontrer que les compositions d'automates sont correctes, c'est-à-dire que $\mathcal{L}_{\text{auto}}(A_{\cup}) = L_1 \cup L_2$, $\mathcal{L}_{\text{auto}}(A_{\cdot}) = L_1 \cdot L_2$, $\mathcal{L}_{\text{auto}}(A_{*}) = L_1^*$ et $\mathcal{L}_{\text{auto}}(A_{\cap}) = L_1 \cap L_2$.

1. Démontrer que l'automate $A_{\cup} = (Q^{\cup}, \Sigma, q_{\text{init}}^{\cup}, \Delta^{\cup}, F^{\cup})$ reconnaît $L_1 \cup L_2$.

2. Démontrer que l'automate $A_+ = (Q^+, \Sigma, q_{\text{init}}, \Delta^+, F)$ reconnaît $L_1 \cdot L_2$.
3. Démontrer que l'automate $A_* = (Q^*, \Sigma, q_{\text{init}}^*, \Delta^*, F^*)$ reconnaît L_1^* .
4. Démontrer que l'automate $A_\cap = (Q^\cap, \Sigma, q_{\text{init}}^\cap, \Delta^\cap, F^\cap)$ reconnaît $L_1 \cap L_2$.

Exercice 99 (♠) — Mot miroir, langage miroir

Soit Σ un alphabet. Nous considérons la définition du mot miroir, définition 121 (p. 170).

1. Donner quelques exemples de mots miroirs.
2. Donner quelques exemples de langages miroirs.

Exercice 100 (♠♠) — Mot miroir, définition inductive

Soit Σ un alphabet. Nous avons défini l'image miroir ou mot miroir d'un mot dans la définition 121 (p. 170) en utilisant la définition de mot sous forme d'application.

1. Donner une définition inductive de l'opérateur $.^R : \Sigma^* \rightarrow \Sigma^*$ en utilisant la définition inductive des mots, définition 54 (p. 30).

Exercice 101 (♠♠) — Miroir de deux mots concaténés

Soit Σ un alphabet. Soient w et w' deux mots sur Σ .

1. Exprimer $(w \cdot w')^R$ en fonction de w^R et w'^R .
2. Démontrer le résultat trouvé à la question précédente.

Exercice 102 (♠♠♠) — Le miroir d'un langage à états est un langage à états

Soient Σ un alphabet et L un langage à états.

1. Démontrer que $L^R = \{u^R \mid u \in L\}$, le langage miroir de L , est un langage à états.

Exercice 103 (♠♠♠) — Image par morphisme

Soient Σ et Σ' deux alphabets. Soit $h : \Sigma^* \rightarrow \Sigma'$ un morphisme de mots entre Σ et Σ' .

Soit A un AFED sur Σ reconnaissant un langage L .

1. Définir un ϵ -AFEND sur Σ' reconnaissant le langage $h(L)$.
2. Démontrer que l'automate obtenu à la question précédente reconnaît bien $h(L)$.

Exercice 104 (♠♠♠) — Image par morphisme inverse

Soient Σ et Σ' deux alphabets. Soit $h : \Sigma^* \rightarrow \Sigma'$ un morphisme de mots entre Σ et Σ' .

Soit A un AFED sur Σ reconnaissant un langage L .

1. Définir un ϵ -AFEND sur Σ' reconnaissant le langage $h^{-1}(L)$.
2. Démontrer que l'automate obtenu à la question précédente reconnaît bien $h^{-1}(L)$.

8.3.5 Algorithmes sur les ϵ -AFEND

Exercice 105 (♠♦) — Algorithme pour calculer l' ϵ -fermeture d'un état

Soit Σ un alphabet.

1. Donner un algorithme qui calcule l' ϵ -fermeture d'un état d'un ϵ -AFEND passé en paramètre.

Exercice 106 (♠♦) — Algorithme pour déterminiser les ϵ -AFEND

1. En utilisant le résultat de l'exercice 105, adapter l'algorithme 32, trouvé en solution de l'exercice 79 pour la déterminisation des AFEND, pour trouver un algorithme de déterminisation des ϵ -AFEND.

Exercice 107 (♠♦♦) — Algorithme pour déterminer le langage universel

Soient Σ et Σ' deux alphabets. Donner des algorithmes qui résolvent les problèmes suivants.

1. Déterminer si un ϵ -AFEND sur Σ reconnaît le langage universel sur Σ .
2. Déterminer si un ϵ -AFEND sur Σ reconnaît au moins le langage universel sur Σ' .

8.4 Indications pour résoudre les exercices

Indications pour l'exercice 84 (p. 174)

1. Il y a deux « chemins » dans l'automate pour distinguer les mots avec une partie entière et une partie décimale vide.
2. Raffiner l'automate trouvé à la question précédente, en distinguant le symbole 0, lors de la réception du premier symbole.

Indications pour l'exercice 82 (p. 172)

1. Définir la fonction par induction sur les mots, en distinguant le cas du mot vide, le cas d'un symbole et le cas d'un mot non vide.

Indications pour l'exercice 85 (p. 174)

1. Trouver une transformation sur les mots permettant de passer de la convention gros-boutiste vers la convention petit-boutiste.

Indications pour l'exercice 86 (p. 174)

1. Trouver un automate qui reconnaît $Suf(L)$ à partir de l'automate reconnaissant L . Utiliser la notion de coaccessibilité.

2. À partir de la définition de préfixe, suffixe et facteur, définition 58 (p. 30),

Indications pour l'exercice 87 (p. 174)

1. Partir simultanément de l'état initial en suivant la relation de transition et d'un état accepteur en suivant la relation de transition inverse, pas à pas. Arriver à un « état de rencontre ».

Indications pour l'exercice 88 (p. 175)

1. Démontrer l'inclusion des langages L^* et $L^* \cdot L^*$ l'un dans l'autre. Pour la direction $L^* \subseteq L^* \cdot L^*$, démontrer que tout mot de L^* peut s'écrire comme un mot de $L^* \cdot L^*$. Pour la direction $L^* \cdot L^* \subseteq L^*$, démontrer que pour tout $u, v \in L^*$, $u \cdot v \in L^*$, par induction sur u

Indications pour l'exercice 89 (p. 175)

1. Utiliser (et démontrer) que, pour tout langage L , $L \subseteq L^*$. Démontrer l'inclusion de L^* et $(L^*)^*$ l'un dans l'autre. $L^* \subseteq (L^*)^*$ est immédiat. $L^* \supseteq (L^*)^*$ se démontre par induction en utilisant la définition de la fermeture de Kleene et en utilisant le résultat de l'exercice 88.

Indications pour l'exercice 90 (p. 175)

1. Utiliser une induction sur les mots de L_1^* (en utilisant les règles de construction de L_1^*).
2. C'est une conséquence du résultat de la question précédente.
3. Utiliser par exemple des langages contenant chacun un mot de longueur 1 différents.
4. Utiliser des langages tels que l'un contienne un mot et l'autre contient des mots qui permettent de construire le mot du premier langage.

Indications pour l'exercice 91 (p. 175)

1. Démontrer l'inclusion des langages l'un dans l'autre. Pour chacune des directions, utiliser la définition inductive de la fermeture de Kleene d'un langage.
2. Considérer des langages d'intersection vide.
3. La condition nécessaire et suffisante utilise le mot vide ϵ .

Indications pour l'exercice 96 (p. 177)

1. Introduire un nouvel état initial.
2. Examiner les états prédécesseurs des états accepteurs.
3. Suivre le même principe que pour la question précédente et introduire un nouvel état accepteur.

4. *Duplicer l'automate.*

Indications pour l'exercice 98 (p. 177)

1. *Raisonner sur l'évolution des configurations qui doit suivre la méthode de composition des automates.*

Indications pour l'exercice 101 (p. 178)

1. *Utiliser quelques mots pour trouver le résultat.*
2. *La preuve en utilisant la définition de mots sous forme d'application est immédiate. La preuve en utilisant la définition inductive de mots peut se faire en s'appuyant sur l'exercice 100.*

Indications pour l'exercice 102 (p. 178)

1. *Se rammener aux exécutions, à la définition de la fonction miroir et du langage généré à partir de cette fonction.*

Indications pour l'exercice 103 (p. 178)

- 2 *Passer par les automates et utiliser les correspondances entre les fermetures des relations de transitions.*

Indications pour l'exercice 104 (p. 178)

2. *Suivre le même principe que celui de l'exercice 103.*

Indications pour l'exercice 105 (p. 179)

1. *Utiliser la définition inductive de l' ϵ -fermeture d'un état. Itérer sur un ensemble des états contenant l' ϵ -fermeture et ajouter les états reliés par une ϵ -transition jusqu'à qu'aucun état ne soit ajouté à cet ensemble.*

Indications pour l'exercice 107 (p. 179)

1. *Utiliser l'une des trois solutions possibles suivantes :*
 - (i) *l'automate complémentaire définition 79 (p. 72) et le test du langage vide ;*
 - (ii) *la distinguabilité entre états algorithme 30 (p. 123) ;*
 - (iii) *l'algorithme de minimisation algorithme 31 (p. 124) et définition 103 (p. 125).*
2. *Distinguer trois cas : $\Sigma' \setminus \Sigma = \emptyset$, $\Sigma = \Sigma'$ et les autres cas.*

8.5 Solutions des exercices

Solution de l'exercice 81 (page 172)

1. Le mot abb est accepté par cet automate. L'exécution de l'automate sur abb est représentée sur la figure 8.2a. Nous observons que l'une des exécutions de l'automate termine dans la configuration acceptante $(2, \epsilon)$.
2. Le mot ac est rejeté par cet automate. L'exécution de l'automate sur ac est représentée sur la figure 8.2b. Nous observons qu'aucune des exécutions de l'automate termine dans une configuration acceptante.

Solution de l'exercice 82 (page 172)

1. La fonction de transition est définie comme suit :
 - $\Delta^*(q, \epsilon) = \text{e-fermeture}(q)$;
 - $\Delta^*(q, a) = \bigcup_{q' \in \text{e-fermeture}(q)} \text{e-fermeture}(\{q'' \mid (q', a, q'') \in \delta\})$;
 - $\Delta^*(q, a \cdot x) = \bigcup_{q_a \in Q_a} \Delta^*(q_a, x)$, avec $Q_a = \Delta^*(q, a)$.
2. La fonction de transition est définie comme suit : $\Delta^*(Q, w) = \bigcup_{q \in Q} \Delta^*(q, w)$.

8.5.1 Trouver un automate

Solution de l'exercice 83 (page 173)

1. Soit $w = a_1 \cdots a_n$ le mot à reconnaître, l'automate est :

$$(\{q, q', q_1, \dots, q_{n-1}\}, \Sigma, q, \delta, \{q'\})$$

où δ est l'ensemble des transitions $\{(q, a_1, q_1), (q_1, a_2, q_2), \dots, (q_{n-1}, a_n, q')\}$.

Solution de l'exercice 84 (page 174)

1. L'automate est donné dans la figure 8.3. À partir de l'état initial (0), l'automate peut lire le symbole + ou - ou encore ϵ (rendant le symbole du signe optionnel) pour aller dans l'état 1. À partir de l'état 1, l'automate peut lire un symbole parmi 0, 1, ..., 9 et rester dans l'état 1, lire • et aller dans l'état 2, ou lire un symbole parmi 0, 1, ..., 9 et aller dans l'état 4. Ces deux dernières possibilités sont associées aux deux chemins possibles dans l'automate. La premier force la partie décimale du nombre à être non vide et sera utilisé pour les nombres décimaux avec une partie décimale non vide (en particulier si la partie entière est vide). Le second force la partie entière du nombre à être non vide et sera utilisé pour les nombres décimaux avec une partie entière non vide (en particulier si la partie décimale est vide). Les deux chemins mènent à l'état 3 où l'automate peut lire un symbole parmi 0, 1, ..., 9 et rester dans l'état 3 ou aller à l'état 5 par ϵ . Notons que lorsque la partie entière et décimale sont vides, les deux chemins peuvent être suivis dans l'automate.
2. L'automate trouvé à la question précédente est modifié comme suit : différentes situations correspondant à l'état 1 dans le premier automate sont maintenant distinguées (états 1, 1' et 1'' dans le nouvel automate) et l'état 4 est supprimé. On distingue le

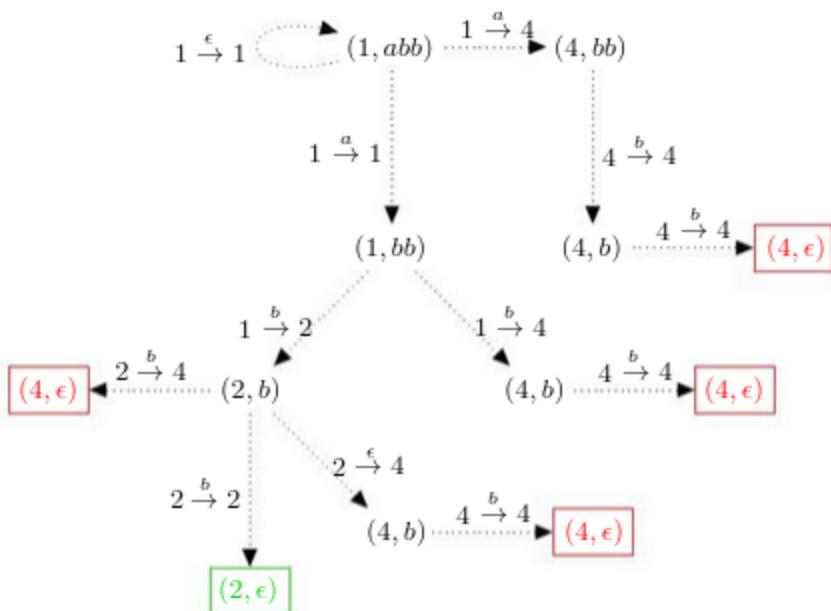
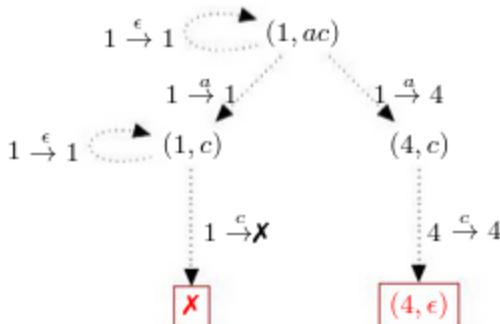
(a) Représentation schématique des exécutions du mot abb – mot accepté par l’automate.(b) Représentation schématique des exécutions du mot ac – mot rejeté par l’automate.

Figure 8.2 – Exécutions de l’automate de la figure 8.1b sur deux mots : abb (figure 8.2a) et ac figure 8.2b. Chaque exécution est représentée sous forme d’arbre dont les nœuds sont des configurations, les feuilles (encadrées) des configurations terminales, en vert les configurations acceptantes, en rouge les configurations non acceptantes. Chaque arête est étiquetée par la transition justifiant l’évolution de la configuration correspondante.

premier symbole reçu, si celui-ci est 0, l’automate se rend dans l’état $1''$ où après avoir lu \bullet , l’automate se rend dans l’état 2 sinon, si celui-ci est un symbole parmi $1, \dots, 9$, l’automate se rend dans l’état $1'$. À partir de l’état 1 il est aussi possible de ne pas avoir de partie entière et l’automate se rend également dans l’état $1''$ sur ϵ . À partir de l’état $1'$, l’automate peut lire un symbole $0, 1, \dots, 9$ et rester dans l’état $1'$ ou de manière non déterministe aller dans l’état 2 ou 3 sur le symbole \bullet pour lire une partie décimale non vide ou vide respectivement.

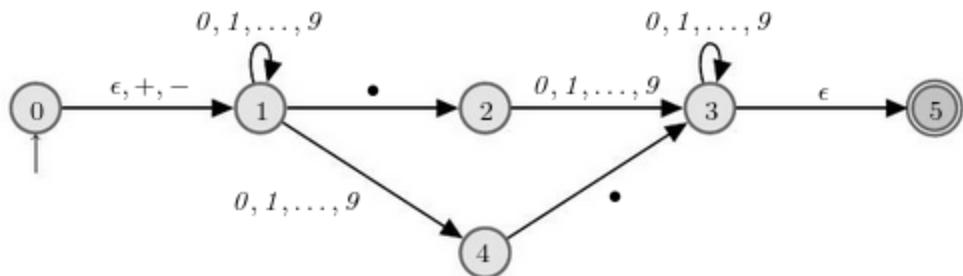
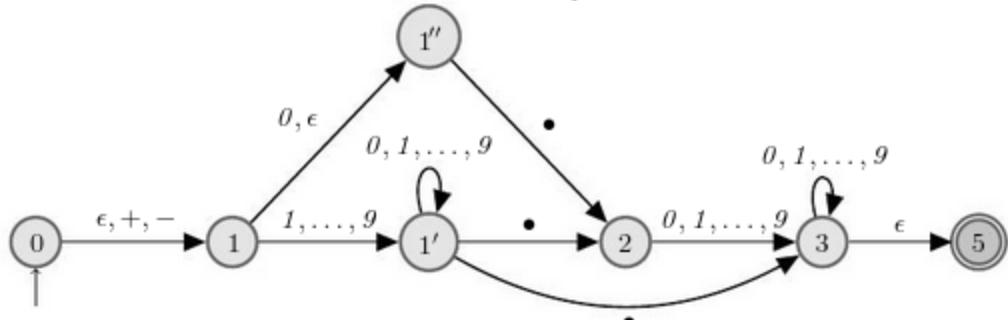

 (a) ϵ -AFEND solution de la question 1.

 (b) ϵ -AFEND solution de la question 2.

 Figure 8.3 – ϵ -AFEND solution de l'exercice 84.

Solution de l'exercice 85 (page 174)

1. Nous observons qu'un entier en convention gros-boutiste est le miroir d'un entier en convention petit-boutiste. Ainsi, nous partons de l'automate trouvé dans l'exercice 22, nous appliquons la transformation miroir et nous le déterminisons puis minimisons. L'automate est donné dans la figure 8.4. Il se trouve que nous trouvons le même automate (où cette fois ci les états ne correspondent pas à des restes de la division euclidienne par 3).

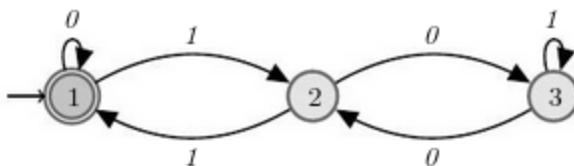


Figure 8.4 – Automate solution de l'exercice 85.

Solution de l'exercice 86 (page 174)

Comme L est un langage à états, il existe un automate déterministe A_L qui reconnaît L . Supposons, sans perte de généralité, que A_L est minimal et avec tous ses états accessibles et coaccessibles. Soit $A_L = (Q^L, \Sigma, q_{\text{init}}^L, \delta^L, F^L)$.

- Nous construisons un automate non déterministe avec ϵ -transitions $A_{\text{Suf}(L)}$ qui reconnaît $\text{Suf}(L)$. Intuitivement, pour trouver l'automate, nous faisons les deux observations suivantes :
 - En examinant la définition de $\text{Suf}(L)$, les mots à reconnaître sont ceux qui “terminent” les mots dans L .
 - Un mot accepté par un automate a son exécution qui démarre de l'état initial et termine dans sa dernière configuration dans un état accepteur.

Ainsi, un mot reconnu par $A_{\text{Suf}(L)}$ doit permettre d'atteindre un état accepteur à partir de son état initial si et seulement si il permet d'atteindre un état accepteur de A_L à partir d'un état coaccessible. Nous en déduisons que pour obtenir $A_{\text{Suf}(L)}$, il nous suffit de pouvoir considérer n'importe quel état coaccessible de A_L comme état initial. Pour cela, nous ajoutons une ϵ -transition depuis l'état initial de A_L vers tous ses états coaccessibles (c'est-à-dire tous ses états car nous avons supposé que tous ses états étaient coaccessibles). Ainsi, l'automate $A_{\text{Suf}(L)}$ est $(Q^L, \Sigma, q_{\text{init}}^L, \delta^L \cup (\{q_{\text{init}}^L\} \times \{\epsilon\}) \times Q^L), F^L$.

- En utilisant un raisonnement similaire à celui de la question précédente et celui suivi dans l'exercice 54, nous construisons un automate non déterministe avec ϵ -transitions $A_{\text{Fact}(L)}$ qui reconnaît $\text{Fact}(L)$. Cet automate est tel qu'un mot peut être accepté en partant de n'importe quel état coaccessible et en s'arrêtant dans n'importe quel autre état coaccessible en suivant les transitions initiales de la fonction de transition de A_L . Ainsi, l'automate $A_{\text{Fact}(L)}$ est $(Q^L, \Sigma, q_{\text{init}}^L, \delta^L \cup (\{q_{\text{init}}^L\} \times \{\epsilon\}) \times Q^L), Q^L$.

Solution de l'exercice 87 (page 174)

- Nous définissons l'AFED $A^{\frac{1}{2}} = ((Q \times Q) \cup \{q'_{\text{init}}\}, \Sigma, q'_{\text{init}}, \Delta', F')$ avec :
 - $F' = \{(q, q) \mid q \in Q\}$,
 - $q'_{\text{init}} \notin Q$ et
 - $\Delta' = \{((p_1, p_2), a, (q_1, q_2)) \mid (p_1, a, q_1) \in \Delta \wedge \exists b \in \Sigma. (q_2, b, p_2) \in \Delta\}$
 $\quad \cup \{(q'_{\text{init}}, \epsilon, (q_{\text{init}}, q_f)) \mid q_f \in F\}$
- Nous réutilisons les notations introduites dans la solution de l'exercice 68 (p. 147), en particulier la relation de transition miroir $R(\Delta) = \{(q, s, p) \mid (p, s, q) \in \Delta\}$. De plus, nous utilisons la propriété suivante :

$$\forall q, q' \in Q. \forall w \in \Sigma^*. \forall n \in \mathbb{N}. (q, w, q') \in T(n, \Delta) \iff (q', w^R, q) \in T(n, R(\Delta)).$$

Pour démontrer que le langage reconnu par cet automate est bien le langage attendu, nous utilisons la propriété suivante :

$$\begin{aligned} & ((p_1, p_2), w, (q_1, q_2)) \in T(n, \Delta') \\ & \text{ssi} \\ & \left\{ \begin{array}{l} (p_1, w, q_1) \in T(n, \Delta) \\ \wedge \exists w' \in \Sigma^*. (|w'| = |w| \wedge (p_2, w', q_2) \in T(n, R(\Delta))) \end{array} \right. \end{aligned}$$

La propriété se montre par récurrence sur $n \in \mathbb{N}^*$.

- Tous les mots reconnus par $A^{\frac{1}{2}}$ sont dans $L^{\frac{1}{2}}$. Considérons un mot w reconnu par $A^{\frac{1}{2}}$. Il existe deux états q_f et q tels l'exécution sur w existe : $(q'_\text{init}, w) \cdot ((q_\text{init}, q_f), w) \cdots ((q, q), \epsilon)$.

$$\exists n \in \mathbb{N}^*. \exists w' \in \Sigma^*. \left\{ \begin{array}{l} (q_\text{init}, w, q) \in T(n, \Delta) \\ (q_f, w', q) \in T(n, R(\Delta)) \end{array} \right.$$

De plus, en utilisant la propriété intermédiaire $(q_f, w', q) \in T(n, R(\Delta))$ si et seulement si $(q, R(w'), q_f) \in T(n, \Delta)$. En utilisant $(q_0, w, q) \in T(n, \Delta)$ and $(q, R(w'), q_f) \in T(n, \Delta)$, nous obtenons $(q_\text{init}, w \cdot R(w'), q_f) \in T(2 \times n, \Delta)$. De plus, $w \cdot R(w') \in L$ implique $w \in L^{\frac{1}{2}}$ (car $|R(w')| = |w|$).

- Tous les mots de $L^{\frac{1}{2}}$ sont reconnus par $A^{\frac{1}{2}}$. Considérons $u \in L^{\frac{1}{2}}$. Il existe $v \in \Sigma^*$ tel que $|u| = |v|$ et $u \cdot v \in L$. De plus, $|u| = |v|$ et $u \cdot v \in L$ impliquent qu'il existe un état q tel que $(q_\text{init}, u, q) \in T(n, \Delta)$ et $(q, v, q_f) \in T(n, \Delta)$ avec $q_f \in F$. Nous avons $(q, v, q_f) \in T(n, \Delta)$ si et seulement si $(q_f, R(v), q) \in T(n, R(\Delta))$.

8.5.2 Fermeture de Kleene

Solution de l'exercice 88 (page 175)

1. Soient Σ un alphabet et $L \subseteq \Sigma^*$ un langage sur Σ . Montrons l'égalité de L^* et $L \cdot L^*$ en montrant l'inclusion des deux langages l'un dans l'autre.
 - Preuve de $L^* \subseteq L^* \cdot L^*$. Par définition de la fermeture de Kleene d'un langage, $\epsilon \in L^*$. De plus, tout mot $u \in \Sigma^*$ peut s'écrire $u \cdot \epsilon$, où ϵ est le mot vide sur l'alphabet Σ .
 - Preuve de $L^* \cdot L^* \subseteq L^*$. Pour cela, nous démontrons que $\forall u, v \in L^*, u \cdot v \in L^*$, par induction sur u .
 - Cas de base : $u = \epsilon$. Soit $v \in L^*$, on a $v \cdot \epsilon = v \in L^*$, par définition de l'opérateur de concaténation appliqué avec le mot vide.
 - Pas d'induction. Soit $u \in L^*$. Supposons la propriété vérifiée pour tous les mots de longueur inférieure à celle de u et qui sont des mots de L^* , c'est-à-dire que pour tout $u' \prec u$ et $v \in L^*$, si $u' \in L^*$, alors $u' \cdot v \in L^*$. Soit $a \in \Sigma$ un symbole, considérons le mot $u \cdot a$. Soit $v \in L^*$, démontrons que $u \cdot a \cdot v \in L^*$. Le mot $u \cdot a$ est différent de ϵ et appartient à L^* , peut donc s'écrire sous la forme $u \cdot a = u' \cdot u''$ avec $u' \in L^*$ et $u'' \in L$, par définition de la fermeture de Kleene de L (au pire des cas u' est ϵ).
 - Cas $u' = \epsilon$. Nous déduisons le résultat immédiatement en utilisant la définition de la fermeture de Kleene car $u'' \in L$ et $v \in L^*$.
 - Cas $u' \neq \epsilon$. Par associativité de la concaténation, $(u' \cdot u'') \cdot v = u' \cdot (u'' \cdot v)$. Comme $u'' \in L$ et $v \in L^*$, comme pour le cas précédent $u'' \cdot v \in L^*$. En utilisant l'hypothèse d'induction, comme u' est de longueur inférieure ou égale à celle de u , nous déduisons que $u' \cdot (u'' \cdot v) \in L^*$.

Dans tous les cas, nous obtenons le résultat attendu.

Solution de l'exercice 89 (page 175)

1. Intuitivement, l'idempotence de la fermeture de Kleene indique que, étant donné un langage L , le langage résultant d'une application de la fermeture de Kleene, et les langages résultant de deux applications successives de la fermeture de Kleene à ce langage, sont tous les mêmes. Notons, que la propriété d'idempotence de la fermeture de Kleene n'est pas restreinte aux langages réguliers. Nous utiliserons également le fait que $L \subseteq L^*$, pour tout langage $L \subseteq L^*$. Ceci se montre en considérant un mot $u \in L$ et en utilisant la deuxième règle de construction de la fermeture de Kleene d'un langage : comme $u \cdot \epsilon = u$ et $\epsilon \in L^*$, on obtient $u \in L^*$.

Soient Σ un alphabet et $L \subseteq \Sigma^*$ un langage sur Σ . Considérons les langages suivants :

- L^* , la fermeture de Kleene de L ;
- $(L^*)^*$, la fermeture de Kleene de L^* .

Démontrons que $L^* = (L^*)^*$, en montrant l'inclusion des deux ensembles l'un dans l'autre.

- Preuve de $L^* \subseteq (L^*)^*$. Ceci est une conséquence immédiate de la propriété $L \subseteq L^*$, pour tout langage L et utilisée avec le langage L^* .
- Preuve de $(L^*)^* \subseteq L^*$. Nous le démontrons par induction en utilisant la définition de la fermeture de Kleene de L^* .
 - Le mot ϵ appartient par définition de la fermeture de Kleene de tout langage et en particulier à celle de L .
 - Soit $v \in (L^*)^*$, supposons que $v \in L^*$. Soit $u \in L^*$, nous avons $u \cdot v \in L^* \cdot L^*$ et en utilisant le résultat de l'exercice 88, $u \cdot v \in L^*$.

Solution de l'exercice 90 (page 175)

1. Nous démontrons que si $L_1 \subseteq L_2$, alors $L_1^* \subseteq L_2^*$ par induction sur les mots de L_1^* (en utilisant les règles de construction de L_1^*).
 - Considérons le mot $\epsilon \in L_1^*$. Par définition de la fermeture de Kleene d'un langage, le mot ϵ fait toujours partie de la fermeture de Kleene. Ainsi, $\epsilon \in L_2^*$.
 - Soit $v_1 \in L_1^*$ et supposons que $v_1 \in L_2^*$ (hypothèse d'induction). Considérons le mot $u_1 \cdot v_1$ avec $u_1 \in L_1$. Comme $L_1 \subseteq L_2$, alors $u_1 \in L_2$. En utilisant la définition de la fermeture de Kleene du langage L_2 , de $u_1 \cdot v_1 \in L_2 \cdot L_2^*$, nous obtenons $u_1 \cdot v_1 \in L_2^*$.
2. Nous obtenons $L_1^* \cup L_2^* \subseteq (L_1 \cup L_2)^*$, immédiatement à partir du résultat de la question précédente et en observant que $L_1 \subseteq L_1 \cup L_2$ et $L_2 \subseteq L_1 \cup L_2$.
3. Pour montrer qu'en général $L_1^* \cup L_2^* \neq (L_1 \cup L_2)^*$, nous donnons un contre-exemple à l'égalité $L_1^* \cup L_2^* = (L_1 \cup L_2)^*$, en prenant $L_1 = \{a\}$ et $L_2 = \{b\}$. Alors, nous avons $L_1^* \cup L_2^* = \{\epsilon, a, a \cdot a, a \cdot a \cdot a, \dots\} \cup \{\epsilon, b, b \cdot b, b \cdot b \cdot b, \dots\}$ alors que $(L_1 \cup L_2)^* = \{\epsilon, a, b, a \cdot b, b \cdot a, a \cdot a, b \cdot b, \dots\}$. Nous observons en particulier que $a \cdot b \in (L_1 \cup L_2)^*$ mais $a \cdot b \notin L_1^* \cup L_2^*$.
4. Nous proposons les langages suivants : $L_1 = \{a \cdot b\}$ et $L_2 = \{a, b\}$. Nous avons bien $L_1 \not\subseteq L_2$, $L_2 \not\subseteq L_1$ et $L_1^* \cup L_2^* = (L_1 \cup L_2)^*$.

Solution de l'exercice 91 (page 175)

1. Rappelons que tout langage est inclus dans sa fermeture de Kleene, c'est-à-dire pour tout langage L , $L \subseteq L^*$. Soient L_1 et L_2 deux langages. Nous démontrons l'égalité entre les deux langages en montrant l'inclusion de l'un dans l'autre. Nous utilisons, pour chacune des directions, la définition inductive de la fermeture de Kleene.

- Preuve de $(L_1 \cup L_2)^* \subseteq (L_1^* \cdot L_2^*)^*$.
 - Cas de base. Le mot ϵ appartient à la fermeture de Kleene de n'importe quel langage.
 - Pas d'induction. Soit $u \cdot v \in (L_1 \cup L_2)^*$ avec $u \in L_1 \cup L_2$ et $v \in (L_1 \cup L_2)^*$. Supposons que $v \in (L_1^* \cdot L_2^*)^*$ (hypothèse d'induction). Nous distinguons deux cas :
 - Cas $u \in L_1$. Alors $u \in L_1^*$. De plus, le mot u s'écrit $u = u \cdot \epsilon$ et ainsi nous déduisons successivement $u \in L_1^* \cdot L_2^*$ (car $\epsilon \in L_2^*$) puis $u \in (L_1^* \cdot L_2^*)^*$.
 - Cas $u \in L_2$. En suivant le même raisonnement, nous déduisons que $u \in (L_1^* \cdot L_2^*)^*$.
 - Preuve de $(L_1^* \cdot L_2^*)^* \subseteq (L_1 \cup L_2)^*$.
 - Cas de base. Le mot ϵ appartient à la fermeture de Kleene de n'importe quel langage.
 - Pas d'induction. Soit $u \cdot v \in (L_1^* \cdot L_2^*)^*$ avec $u \in L_1^* \cdot L_2^*$ et $v \in (L_1^* \cdot L_2^*)^*$. Supposons que $v \in (L_1 \cup L_2)^*$ (hypothèse d'induction). Comme $u \in L_1^* \cdot L_2^*$, le mot u s'écrit $u = u_1 \cdot u_2$ avec $u_1 \in L_1^*$ et $u_2 \in L_2^*$. Comme $u_1 \in L_1^*$ et $L_1 \subseteq L_1 \cup L_2$, en utilisant le résultat de l'exercice 90, nous obtenons $u_1 \in (L_1 \cup L_2)^*$. De manière similaire, de $u_2 \in L_2^*$, nous obtenons $u_2 \in (L_1 \cup L_2)^*$. Ainsi, $u = u_1 \cdot u_2 \in (L_1 \cup L_2)^*$. En utilisant le résultat de l'exercice 88, de $u \cdot v \in (L_1 \cup L_2)^* \cdot (L_1 \cup L_2)^*$, nous déduisons que $u \cdot v \in (L_1 \cup L_2)^*$.
- 2. Non. Un contre-exemple est $L_1 = \{a\}$ et $L_2 = \{b\}$. Nous avons $b \in (L_1 \cup L_2)^*$ mais $b \notin (L_1 \cdot L_2)^*$.
- 3. Une condition nécessaire et suffisante pour que $(L_1 \cup L_2)^* = (L_1 \cdot L_2)^*$ est $(\epsilon \in L_1 \wedge \epsilon \in L_2)$.
 - Démontrons que $\epsilon \in L_1 \wedge \epsilon \in L_2$ est une condition suffisante. Démontrons que $(\epsilon \in L_1 \wedge \epsilon \in L_2) \implies (L_1 \cup L_2)^* = (L_1 \cdot L_2)^*$.
 - Démontrons que $(L_1 \cup L_2)^* \subseteq (L_1 \cdot L_2)^*$. Soit $u \in (L_1 \cup L_2)^*$, démontrons que $u \in (L_1 \cdot L_2)^*$. Le mot u peut s'écrire/se décomposer en $u_1 \cdots u_n$ avec $\forall i \in [1, n], u_i \in L_1 \cup L_2$, pour un certain $n \in \mathbb{N}$. c'est-à-dire que chaque facteur u_i est tel que $u_i \in L_1$ ou $u_i \in L_2$. Pour chacun de ces facteurs u_i , nous avons deux cas possibles. Dans le cas où $u_i \in L_1$, alors $u_i \in L_1 \cdot L_2$ car $\epsilon \in L_2$. Dans le cas où $u_i \in L_2$, alors $u_i \in L_1 \cdot L_2$ car $\epsilon \in L_1$.
 - Démontrons que $(L_1 \cup L_2)^* \supseteq (L_1 \cdot L_2)^*$. Soit $u \in (L_1 \cdot L_2)^*$, démontrons que $u \in (L_1 \cup L_2)^*$. Le mot u peut s'écrire/se décomposer en $u_1 \cdots u_n$ avec $\forall i \in [1, n], u_i \in L_1 \cdot L_2$, pour un certain $n \in \mathbb{N}$. Chaque facteur u_i peut

se décomposer en $u_i^1 \cdot u_i^2$, avec $u_i^1 \in L_1$ et $u_i^2 \in L_2$. Ainsi, le mot u peut s'écrire $u'_1 \cdots u'_{2 \times n}$, avec $\forall i \in [1, 2 \times n]$. $u'_i \in L_1 \vee u'_i \in L_2$, c'est-à-dire $\forall i \in [1, 2 \times n]$. $u'_i \in L_1 \cup L_2$.

- Démontrons que $\epsilon \in L_1 \wedge \epsilon \in L_2$ est une condition nécessaire. Démontrons que $(L_1 \cup L_2)^* = (L_1 \cdot L_2)^* \implies (\epsilon \in L_1 \wedge \epsilon \in L_2)$. Pour cela, nous démontrons la contraposée, c'est-à-dire $(\epsilon \notin L_1 \vee \epsilon \notin L_2) \implies (L_1 \cup L_2)^* \neq (L_1 \cdot L_2)^*$. Prenons le cas où $\epsilon \notin L_1$. Soit u_2 l'un des mots de L_2 avec la longueur la plus petite. Alors $u_2 \notin L_1 \cdot L_2$. En effet, même si u_2 avait un préfixe dans L_1 , celui-ci serait de longueur non nulle et il ne serait pas possible de trouver un suffixe (strict) dans L_2 car il n'y a pas de mot de longueur plus petite dans L_2 . Dans le cas où $\epsilon \notin L_2$, on considère u_1 l'un des mots de L_1 avec la longueur la plus petite et on montre de manière similaire que $u_1 \notin L_1 \cdot L_2$.

8.5.3 Élimination des ϵ -transitions et déterminisation

Solution de l'exercice 92 (page 176)

1. Après éliminations des ϵ -transitions de l' ϵ -AFEND représenté dans le tableau 8.2a, nous obtenons l'AFEND représenté dans la figure 8.5a.
2. Après déterminisation de l'AFEND trouvé à la question précédente, nous obtenons l'AFED décrit par le tableau dans la figure 8.5b.
3. Après déterminisation et élimination des ϵ -transitions « à la volée » de l' ϵ -AFEND dans le tableau 8.2a, nous obtenons l'AFED représenté par le tableau dans la figure 8.5c.

	$\downarrow 0$	1	2*	3	4*
a	2, 4	2, 4		4	
b			1, 3		3

(a) AFEND obtenu après élimination des ϵ -transitions de l' ϵ -AFEND représenté dans le tableau 8.2a.

	$\downarrow \{0\}$	$\{2, 4\}^*$	$\{1, 3\}$	\emptyset
a	{2, 4}	\emptyset	{2, 4}	\emptyset
b	\emptyset	{1, 3}	\emptyset	\emptyset

(b) AFED obtenu après déterminisation de l'AFEND dans la figure 8.5a.

	$\downarrow \{0, 1, 3\}$	$\{2, 4\}^*$	$\{1, 3\}$	\emptyset
a	{2, 4}	\emptyset	{2, 4}	\emptyset
b	\emptyset	{1, 3}	\emptyset	\emptyset

(c) AFED obtenu après élimination des ϵ -transitions et déterminisation « à la volée » de l' ϵ -AFEND représenté dans le tableau 8.2a.

	$\downarrow 1$	2*	3	4
a	2	4	1	4
b	4	3	4	4

(d) AFED obtenu après renommage des états des AFED dans les figures 8.5b et 8.5c.

Figure 8.5 – Automates obtenus après élimination des ϵ -transitions et déterminisation dans l'exercice 92 (p. 176). Les automates sont représentés sous forme tabulaire. La flèche indique l'état initial et une étoile indique un état accepteur.

4. Après renommage des états des deux AFED dans les figures 8.5b et 8.5c, nous obtenons l'AFED représenté par le tableau dans la figure 8.5d.

Solution de l'exercice 93 (page 176)

- Après élimination des ϵ -transitions de l' ϵ -AFEND représenté dans le tableau 8.2b, nous obtenons l'AFEND représenté dans la figure 8.6a.
- Après déterminisation de l'AFEND trouvé à la question précédente, nous obtenons l'AFED décrit par le tableau dans la figure 8.6b.
- Après déterminisation et élimination des ϵ -transitions « à la volée » de l' ϵ -AFEND dans le tableau 8.2b, nous obtenons l'AFED représenté par le tableau dans la figure 8.6c.
- Après renommage des états des deux AFED dans les figures 8.6b et 8.6c, nous obtenons l'automate représenté par le tableau dans la figure 8.6d.

	$\downarrow 0$	1	2	3	4^*
a	3, 2	2, 1, 3	2, 1, 3		
b			3, 4		

(a) AFEND obtenu après élimination des ϵ -transitions de l' ϵ -AFEND représenté dans le tableau 8.2b.

	$\downarrow 0$	$\{2, 3\}$	$\{1, 2, 3\}$	$\{3, 4\}^*$	\emptyset
a	$\{2, 3\}$	$\{1, 2, 3\}$	$\{1, 2, 3\}$	\emptyset	\emptyset
b	\emptyset	$\{3, 4\}$	$\{3, 4\}$	\emptyset	\emptyset

(b) AFED obtenu après déterminisation de l'AFEND dans la figure 8.6a.

	$\downarrow \{0, 1, 3\}$	$\{1, 2, 3\}$	$\{1, 2\}$	$\{4\}^*$	\emptyset
a	$\{1, 2, 3\}$	$\{1, 2\}$	$\{1, 2\}$	\emptyset	\emptyset
b	\emptyset	$\{4\}$	$\{4\}$	\emptyset	\emptyset

(c) AFED obtenu après élimination des ϵ -transitions et déterminisation « à la volée » de l' ϵ -AFEND représenté dans le tableau 8.2b.

	$\downarrow 1$	2	3	4^*	5
a	1	3	3	5	5
b	5	4	4	5	5

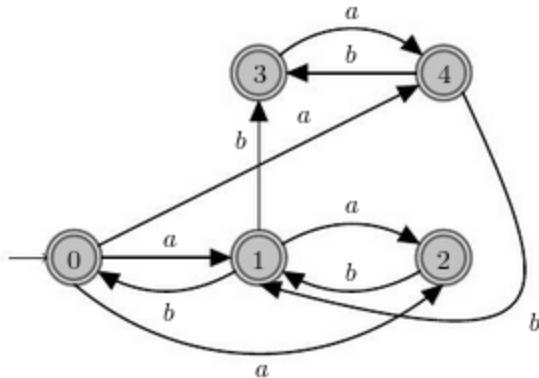
(d) AFED obtenu après renommage des états des AFED dans les figures 8.6b et 8.6c.

Figure 8.6 – Automates obtenus après élimination des ϵ -transitions et déterminisation dans l'exercice 93 (p. 176). Les automates sont représentés sous forme tabulaire. La flèche indique l'état initial et une étoile indique un état accepteur.

Solution de l'exercice 94 (page 176)

Nous considérons l' ϵ -AFEND dans la figure 8.1a défini sur $\Sigma = \{a, b\}$.

- Les mots a et $a \cdot b$ sont acceptés. Tous les mots qui commencent par le symbole b ne sont pas acceptés.
- En éliminant les ϵ -transitions, nous obtenons l'automate représenté en version graphique dans la figure 8.7a et en version tabulaire dans la figure 8.7b.
- En déterminisant l'automate obtenu à la question précédente, nous obtenons l'automate représenté dans la figure 8.7c où nous renommons les états (en vue de la minimisation) pour obtenir l'automate (équivalent) représenté dans la figure 8.7d.
- L'exécution de l'algorithme de minimisation est représenté dans la figure 8.7e. En particulier, nous déduisons de l'exécution de l'algorithme de minimisation que l'automate n'est pas minimal et que les états 0 et 5 peuvent être fusionnés. L'automate minimal est représenté dans la figure 8.7f (où l'état 05 correspond à la fusion des états 0 et 5).



	$\downarrow 0*$	$1*$	$2*$	$3*$	$4*$
a	1, 4, 2	2		4, 2	
b		0, 3	1		1, 3

(a) AFEND en représentation graphique obtenu après élimination des ϵ -transitions de l' ϵ -AFEND dans la figure 8.1a.

	$\downarrow \{0\}*$	$\{2\}*$	$\{0, 1, 3\}*$	$\{1\}*$	$\{1, 4, 2\}*$	$\{0, 3\}*$	\emptyset
a	1, 4, 2	\emptyset	1, 4, 2	2	2	1, 4, 2	\emptyset
b	\emptyset	1	0, 3	0, 3	0, 1, 3	\emptyset	\emptyset

(c) AFED obtenu après déterminisation de l'AFEND dans la figure 8.7b.

	$\downarrow 0*$	$1*$	$2*$	$3*$	$4*$	$5*$	6
a	4	6	4	1	1	4	6
b	6	3	5	5	2	6	6

(d) AFED obtenu après renommage des états de l'AFED dans la figure 8.7c.

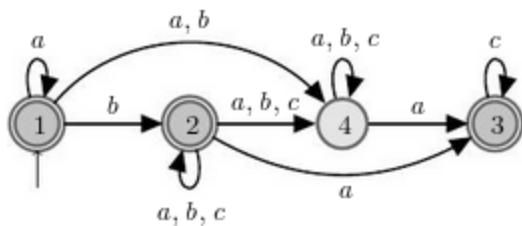
$\equiv 0$	$\equiv 1$	$\equiv 2$	$\equiv 3$
0	1	1	1
1	0	0	0
2	5	5	5
3	2	2	2
4	3	3	3
5	4	4	4
6	6	6	6

(e) Exécution de l'algorithme de minimisation sur l'AFED dans la figure 8.7d.

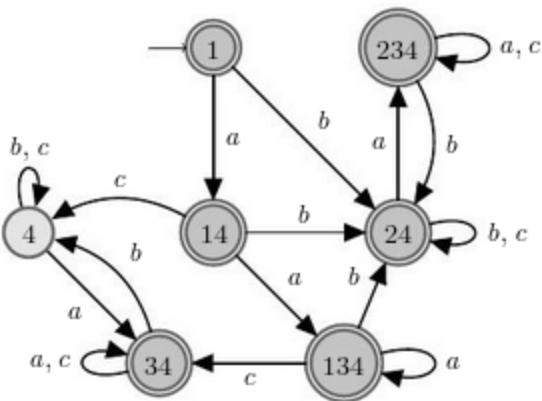
	$\downarrow 05*$	$1*$	$2*$	$3*$	$4*$	6
a	4	6	4	1	1	6
b	6	3	05	05	2	6

(f) AFED minimisé de l'AFED dans la figure 8.7d.

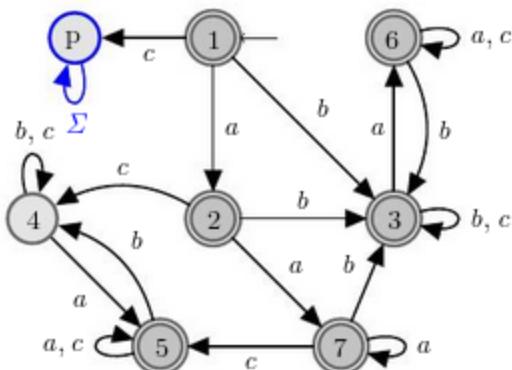
Figure 8.7 – Automates obtenus après élimination des ϵ -transitions et déterminisation dans l'exercice 94 (p. 176).



(a) AFEND obtenu après élimination des ϵ -transitions de l' ϵ -AFEND dans la figure 8.8b.



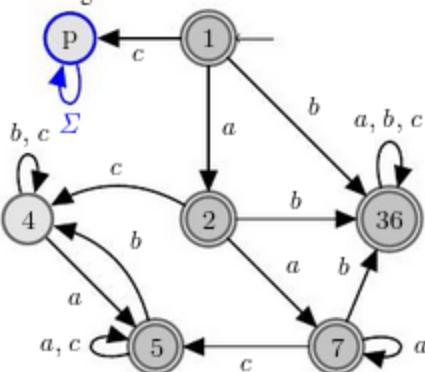
(b) AFED obtenu après déterminisation de l'AFEND dans la figure 8.8a.



(c) Complétion de l'AFED dans la figure 8.8b.

	\equiv_0	\equiv_1	\equiv_2	\equiv_3
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
5	6	6	6	6
6	7	7	7	7
7	5	5	5	5
4	4	4	4	4
p	p	p	p	p

(d) Exécution de l'algorithme de minimisation sur l'AFED dans la figure 8.8c.



(e) AFED minimisé de l'AFED dans la figure 8.8c.

Figure 8.8 – Automates obtenus après élimination des ϵ -transitions et déterminisation dans l'exercice 95 (p. 177).

Solution de l'exercice 95 (page 177)

Nous considérons l' ϵ -AFEND dans la figure 8.1b défini sur $\Sigma = \{a, b, c\}$. Pour les questions suivantes, nous utilisons la représentation graphique des automates.

1. Nous appliquons l'algorithme de suppression des ϵ -transitions et nous obtenons l'automate non déterministe représenté dans la figure 8.8a.
2. Nous appliquons l'algorithme de déterminisation à l'automate de la question précédente et nous obtenons l'automate représenté dans la figure 8.8b.
3. Nous complétons d'abord l'automate et renommons ses états pour obtenir l'automate représenté dans la figure 8.8c. Nous appliquons ensuite l'algorithme de minimisation à l'automate trouvé à la question précédente. L'exécution de l'algorithme est représenté dans la figure 8.8d. Le minimisé est représenté dans la figure 8.8e.

8.5.4 Composition et transformation d' ϵ -AFEND**Solution de l'exercice 96 (page 177)**

Pour chacun des langages considérés, nous montrons comment modifier l'AFED associé au langage pour obtenir un automate reconnaissant le langage décrit dans la question. Nous représentons l'automate ainsi que leur modification de manière schématique. L'automate de départ est représenté dans la figure 8.9a.

Soit $(Q, \Sigma, q_{\text{init}}, \delta, F)$ l'AFED reconnaissant L . L'état q_{init} est l'état initial et un accepteur est ici représenté par un double cercle. L'ellipse englobante représente l'ensemble des états Q .

1. La transformation pour obtenir un automate reconnaissant $\{w \mid s \cdot w \in L\}$ est représentée dans la figure 8.9b. Nous changeons d'état initial dans l'automate en introduisant un nouvel état q'_{init} qui devient l'état initial. Nous sélectionnons ensuite l'état pour lequel il existe une transition depuis l'ancien état initial vers cet état sur le symbole s . Nous ajoutons une transition depuis le nouvel état initial vers cet état. Les transitions de l'automate initial sont conservées. Plus formellement, soit $\delta' = \delta \cup \{(q'_{\text{init}}, \epsilon, q) \mid (q_{\text{init}}, s, q) \in \delta\}$, l'automate transformé est :

$$(Q, \Sigma, q'_{\text{init}}, \delta', F \setminus \{q_{\text{init}}\}).$$

2. La transformation pour obtenir un automate reconnaissant $\{w \mid w \cdot s \in L\}$ est représentée dans la figure 8.9c. Nous sélectionnons tous les états prédécesseurs sur le symbole s de chaque état final sur le symbole s et rendons ces états finaux. Les anciens états finaux ne le sont plus (sauf lorsqu'ils sont prédécesseurs d'un état final). Plus formellement, l'automate transformé est :

$$(Q, \Sigma, q_{\text{init}}, \delta, \{q \in Q \mid \exists (q, s, f) \in \delta, f \in F\}).$$

3. La transformation pour obtenir un automate reconnaissant $\{w \mid w \cdot s \cdot s \in L\}$ est représentée dans la figure 8.9d. Le principe est le même que celui de la transformation précédente. Nous sélectionnons les états accepteurs qui ont une transition entrante sur

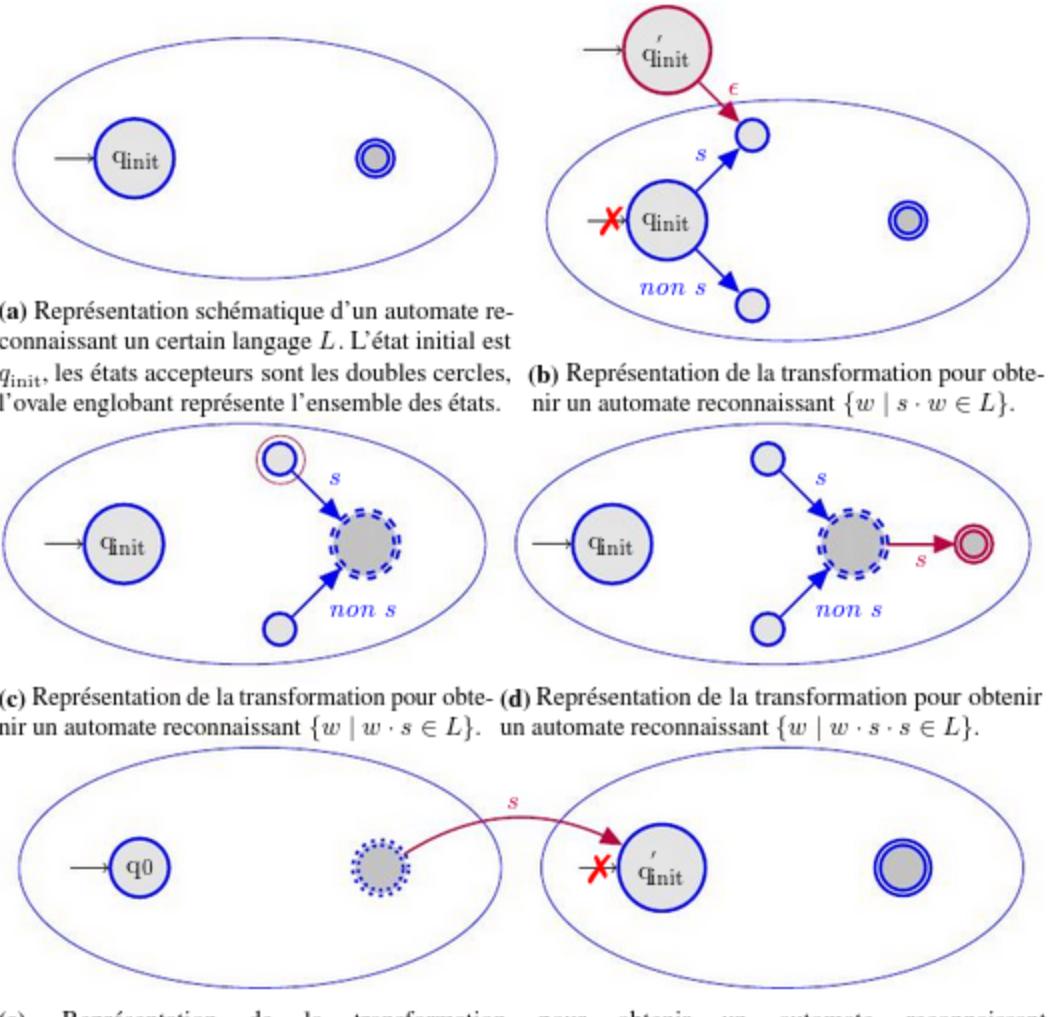


Figure 8.9 – Automates utilisés pour représenter les transformations de l'exercice 96 (p. 177) s'appliquant sur un automate reconnaissant un langage L .

le symbole s . Nous ajoutons un nouvel état à l'automate et ajoutons une transition depuis tous les états accepteurs sélectionnés précédemment vers ce nouvel état sur le symbole s . Ce nouvel état devient l'état final et les anciens finaux ne sont plus finaux. Plus formellement, soient $\text{pre}F = \{q \in Q \mid \exists(q, s, f) \in \delta, f \in F\}$ et $f \notin Q$ un nouvel état, l'automate transformé est :

$$(Q \cup \{f\}, \Sigma, q_{\text{init}}, \delta \cup \text{pre}F \times \{s\} \times \{f\}, \{f\}).$$

4. La transformation pour obtenir un automate reconnaissant $\{w_1 \cdot s \cdot w_2 \mid w_1 \wedge w_2 \in L\}$ est représentée dans la figure 8.9e. Nous dupliquons l'automate (en renommant ses

états). Nous sélectionnons arbitrairement l'un des deux copies de l'automate qui servira à reconnaître la partie w_1 des mots $w_1 \cdot s \cdot w_2$ des mots acceptés. Les états finaux de cet automate ne sont plus finaux. L'autre automate sert à reconnaître la partie w_2 des mots ; l'ancien état initial de cet automate ne l'est plus. Nous ajoutons ensuite une transition sur le symbole s depuis tout ancien état final du premier automate (celui en charge de reconnaître la partie w_1) vers l'ancien état initial du deuxième automate. Plus formellement, soient $(Q^1, \Sigma, q_{\text{init}}^1, \delta^1, F^1)$ et $(Q^2, \Sigma, q_{\text{init}}^2, \delta^2, F^2)$ les deux copies de $(Q, \Sigma, q_{\text{init}}, \delta, F)$, l'automate transformé est :

$$(Q^1 \cup Q^2, \Sigma, q_{\text{init}}^1, \delta^1 \cup \delta^2 \cup \{(f_1, s, q_{\text{init}}^2) \mid f_1 \in F^1\}, F^2).$$

Solution de l'exercice 97 (page 177)

1. $A_{\cup} = (Q^1 \cup Q^2 \cup \{q_{\text{init}}\}, \Sigma, q_{\text{init}}, \Delta^1 \cup \Delta^2 \cup \{(q_{\text{init}}, \epsilon, q_{\text{init}}^1); (q_{\text{init}}, \epsilon, q_{\text{init}}^2)\}, F_1 \cup F_2)$,
2. $A_{\cup} = (Q^1 \cup Q^2, \Sigma, q_{\text{init}}^1, \Delta^1 \cup \Delta^2 \cup \{(q_f^1, \epsilon, q_{\text{init}}^2) \mid q_f^1 \in F_1\}, F_2)$,
3. $A_* = (Q^1 \cup \{q^i, q^f\}, \Sigma, q^i, \Delta^1 \cup \{(q^i, \epsilon, q_{\text{init}}^1), (q_i, \epsilon, q^f)\} \cup \{(f^1, \epsilon, q^f) \mid f^1 \in F^1\}, F^{1'})$, avec $F^{1'}$ qui peut être $\{q^f\}$ ou $\{q^f\} \cup F^1$,
4. Soient $A'_1 = (Q^{1'}, \Sigma, q_{\text{init}}^1, \delta^{1'}, F^{1'})$ et $A'_2 = (Q^{2'}, \Sigma, q_{\text{init}}^2, \delta^{2'}, F^{2'})$ les déterminisés de A_1 et A_2 respectivement, auxquels nous avons appliqué un renommage des états. Nous appliquons la construction de l'automate produit en utilisant les automates A'_1 et A'_2 . Ainsi, nous obtenons :

$$\begin{aligned} A_{\cap} = & (Q^{1'} \times Q^{2'}, \Sigma, (q_{\text{init}}^1, q_{\text{init}}^2), \\ & \{(q^1, q^2), s, (q^{1'}, q^{2'}) \mid (q^1, s, q^{1'}) \in \delta^{1'} \wedge (q^2, s, q^{2'}) \in \delta^{2'}\}, \\ & F^{1'} \times F^{2'}). \end{aligned}$$

Solution de l'exercice 98 (page 177)

Pour démontrer que les compositions sont correctes, nous devons démontrer que les AFEND obtenus (dans l'exercice 97) par ces transformations reconnaissent les langages attendus. Pour cela, nous démontrons l'égalité entre le langage attendu et le langage reconnu, en montrant l'inclusion de l'un dans l'autre.

1. Considérons l'AFEND

$$\begin{aligned} A_{\cup} = & (Q^1 \cup Q^2 \cup \{q_{\text{init}}\}, \Sigma, q_{\text{init}}, \\ & \Delta^1 \cup \Delta^2 \cup \{(q_{\text{init}}, \epsilon, q_{\text{init}}^1), (q_{\text{init}}, \epsilon, q_{\text{init}}^2)\}, \\ & F^1 \cup F^2). \end{aligned}$$

- Preuve de $\mathcal{L}_{\text{auto}}(A_{\cup}) \subseteq L_1 \cup L_2$. Soit u un mot accepté par A_{\cup} . Comme l'ensemble des états accepteurs de A_{\cup} est $F^1 \cup F^2$, l'exécution de u termine dans un état qui est soit dans F^1 soit dans F^2 (rappelons que $F^1 \cap F^2 = \emptyset$), c'est-à-dire dans une configuration qui est soit de la forme (q_f^1, ϵ) avec $q_f^1 \in F^1$ ou de la forme (q_f^2, ϵ) avec $q_f^2 \in F^2$. Comme la relation de transition de A_{\cup} est $\Delta^1 \cup \Delta^2$ (c'est-à-dire qu'elle consiste en uniquement les transitions présentes dans A_1 et A_2 non modifiées, l'exécution de u consiste en une séquence de configurations

obtenues avec la relation de transition correspondante et passe également par l'état initial de l'automate correspondant. Par ailleurs, le passage de la configuration initiale (q_i, u) à la seconde configuration s'obtient soit en prenant la transition $(q_i, \epsilon, q_{\text{init}}^1)$ soit la transition $(q_i, \epsilon, q_{\text{init}}^2)$. En particulier, le passage de configuration ne consomme pas de symbole. Ainsi, si l'exécution de u termine dans un état de F^1 (resp. F^2), alors la seconde configuration de l'exécution est (q_0^1, u) (resp. (q_0^2, u)) et mène à une configuration (q_f^1, ϵ) avec $q_f^1 \in F^1$ (resp. (q_f^2, ϵ) avec $q_f^2 \in F^2$) où l'évolution des configurations est régie par Δ^1 (resp. Δ^2).

- Preuve de $\mathcal{L}_{\text{auto}}(A_{\cup}) \supseteq L_1 \cup L_2$. Soit $u \in L_1 \cup L_2$, le mot u appartient à L_1 ou à L_2 . Par symétrie, supposons que $u \in L_1$ (notons que si u appartient à L_1 et L_2 alors ce cas est traité par les deux cas symétriques mentionnés). Comme $u \in L_1$, u est accepté par A_1 , son exécution sur A_1 termine dans un état accepteur, c'est-à-dire que l'exécution de u sur A_1 peut s'écrire $(q_{\text{init}}^1, u) \cdots (q_f^1, \epsilon)$, avec $q_f^1 \in F_1$. À partir de l'exécution précédente, nous pouvons construire l'exécution $(q_0, u) \cdot (q_{\text{init}}^1, u) \cdots (q_f^1, \epsilon)$ de u sur A_{\cup} où la première dérivation est obtenue en suivant la transition (q_0, ϵ, q_f^1) . Ainsi, comme $q_f^1 \in F_1 \subseteq F_{\cup}$, le mot u est accepté par A_{\cup} .
- 2. Considérons l'AFEND $A_{\cdot} = (Q^1 \cup Q^2, \Sigma, q_{\text{init}}^1, \Delta^1 \cup \Delta^2 \cup \{(q_f^1, \epsilon, q_{\text{init}}^2) \mid q_f^1 \in F^1\}, F^2)$.
 - Preuve de $\mathcal{L}_{\text{auto}}(A_{\cdot}) \subseteq L_1 \cdot L_2$. Soit u un mot accepté par A_{\cdot} . L'exécution de u sur A_{\cdot} termine dans un état de F^2 . Comme d'une part la relation de transition de A_{\cdot} est formée par les transitions dans Δ^1 et Δ^2 qui sont disjointes et d'autre part qu'il existe une transition sur ϵ depuis chaque état de F^1 vers q_0^2 , l'ancien état initial de A_2 , nous en déduisons que l'exécution de u passe par un état de F^1 . Ainsi, l'exécution de u peut se décomposer en trois parties : une partie suivant les transitions de Δ^1 jusqu'à une configuration (q_f^1, u') avec $q_f^1 \in F^1$, une évolution suivant la transition $(q_f^1, \epsilon, q_{\text{init}}^2)$ vers la configuration (q_{init}^2, u') , une évolution vers la configuration (q_2^f, ϵ) avec $q_2^f \in F^2$. Le mot u' est un mot accepté par A_2 (c'est donc un mot de L_2) et est un suffixe de u . Par ailleurs, lors de l'exécution de u , le préfixe de u de longueur $|u| - |u'|$ mène à un état $q_f^1 \in F^1$. Ce préfixe est un mot accepté par A_1 et appartient donc à L_1 . Ainsi, le mot u peut se décomposer en $u_1 \cdot u_2$ (où u_2 est le mot u' référencé précédemment), avec $u_1 \in L_1$ et $u_2 \in L_2$.
 - Preuve de $\mathcal{L}_{\text{auto}}(A_{\cdot}) \supseteq L_1 \cdot L_2$. Soit $u \in L_1 \cdot L_2$. Le mot u peut s'écrire $u_1 \cdot u_2$, avec $u_1 \in L_1$ et $u_2 \in L_2$. Comme u_1 (resp. u_2) est accepté par L_1 (resp. L_2), il existe une exécution de u_1 (resp. u_2) sur A_1 (resp. A_2) qui termine dans un état accepteur de A_1 (resp. A_2). Ces exécutions peuvent s'écrire $(q_{\text{init}}^1, u_1) \cdots (q_f^1, \epsilon)$ et $(q_{\text{init}}^2, u_2) \cdots (q_f^2, \epsilon)$ respectivement avec $q_f^1 \in F^1$ et $q_f^2 \in F^2$. À partir de l'exécution de u_1 sur A_1 (qui est aussi une exécution de u_1 sur A_{\cdot}), nous en déduisons que $(q_0^1, u_1 \cdot u_2) \cdots (q_f^1, u_2)$ est une exécution de $u_1 \cdot u_2$ sur A_1 et également sur A_{\cdot} . Comme $(q_f^1, \epsilon, q_0^2) \in \Delta_{\cdot}$, $(q_0^1, u_1 \cdot u_2) \cdots (q_f^1, u_2) \cdot (q_{\text{init}}^2, u_2)$ est une exécution de $u_1 \cdot u_2$ sur A_{\cdot} . De l'exécution $(q_{\text{init}}^2, u_2) \cdots (q_f^2, \epsilon)$ de u_2 sur

A_2 , et comme $\Delta^2 \subseteq \Delta_*$, nous en déduisons que nous pouvons prolonger l'exécution de $u_1 \cdot u_2$ sur A_* comme suit : $(q_0^1, u_1 \cdot u_2) \cdots (q_f^f, u_2) \cdot (q_{\text{init}}^2, u_2) \cdots (q_2^f, \epsilon)$. Ainsi, comme $q_2^f \in F^2 \subseteq F_*$, $u_1 \cdot u_2$ est un mot accepté par A_* .

3. Considérons l'AFEND $A_* = (Q^1 \cup \{q^i, q^f\}, \Sigma, q^i, \Delta^1 \cup \{(q^i, \epsilon, q_{\text{init}}^1), (q_i, \epsilon, q^f)\} \cup \{(f_1, \epsilon, q^f) \mid f_1 \in F^1\}, \{q^f\})$.

- Preuve de $\mathcal{L}_{\text{auto}}(A_*) \subseteq L_1^*$. Soit u un mot accepté par A_* . Comme $F^* = \{q^f\}$, il existe une exécution de u sur A_* qui termine dans la configuration (q^f, ϵ) . En examinant les transitions dans Δ^* de A_* et sortantes de q^i , il y a deux formes d'exécutions possibles en fonction de la première dérivation appliquée : soit l'exécution de u suit directement la transition (q^i, ϵ, q^f) soit elle suit la transition $(q^i, \epsilon, q_{\text{init}}^1)$.
 - Dans le premier cas, comme il n'y a pas de transition sortante de q^f , le mot u est nécessairement le mot ϵ et $\epsilon \in L_1^*$ car L_1^* est la fermeture de Kleene d'un langage.
 - Dans le second cas, comme il n'y a pas de transition sortant de q^f et qu'il est possible d'atteindre q^f à partir de n'importe quel état de F^1 sur ϵ , l'exécution de u sur A_* peut s'écrire $(q^i, u) \cdot (q_{\text{init}}^1, u) \cdots (q_1^f, \epsilon) \cdot (q^f, \epsilon)$, pour un certain état $q_1^f \in F_1$.

Examinons maintenant la partie suivante de l'exécution $(q_{\text{init}}^1, u) \cdots (q_1^f, \epsilon)$. Comme cette exécution démarre de q_{init}^1 (l'état initial de A_1), cette partie de l'exécution est également une exécution de l'automate A_1 augmenté des transitions depuis les états finaux de A_1 vers l'état initial de A_1 sur ϵ (c'est-à-dire les transitions dans l'ensemble $\{(q_1^f, \epsilon, q_{\text{init}}^1) \mid q_1^f \in F_1\}$). Soit $n \in \mathbb{N}$, le nombre de fois où l'exécution $(q_{\text{init}}^1, u) \cdots (q_1^f, \epsilon)$ utilise une transition dans l'ensemble mentionné précédemment. Nous pouvons décomposer cette exécution en $n + 1$ parties comme suit :

$$\begin{aligned} & (q_{\text{init}}^1, u_0 \cdot u_1 \cdots u_n) \cdots (q_1^{f^0}, u_1 \cdots u_n) \\ & \quad \cdot (q_{\text{init}}^1, u_1 \cdots u_n) \cdots (q_1^{f^i}, u_{i+1} \cdots u_n) \\ & \quad \cdot (q_{\text{init}}^1, u_{i+1} \cdots u_n) \cdots (q_1^{f^n}, \epsilon) \end{aligned}$$

avec $q_1^{f^i}$ pour $i \in [0, n]$ et $q_1^{f^n} = q_1^f$. Ainsi, chaque facteur u_i ($i \in [0, n]$) de u a son exécution démarrant dans l'état initial de A_1 et termine dans un état accepteur de A_1 ; chacun de ces facteurs est donc un mot accepté par A_1 (et donc un mot de L_1). Le mot u est donc la concaténation de mots de L_1 , d'après la définition de la fermeture de Kleene, u appartient à L_1^* .

- Preuve de $\mathcal{L}_{\text{auto}}(A_*) \supseteq L_1^*$. Dans cette direction, nous pouvons suivre le raisonnement précédent en sens inverse. Soit $u \in L_1^*$, comme u appartient à la fermeture de Kleene de L_1 , nous distinguons deux cas : soit u est le mot ϵ soit u est un mot qui s'écrit comme la concaténation de mots de L_1 . Dans le premier cas, nous construisons l'exécution $(q^i, \epsilon) \cdot (q^f, \epsilon)$ en suivant la transition (q^i, ϵ, q^f) . Dans le second cas, nous construisons, comme pour l'autre direction

de la preuve, des exécutions pour chacun des facteurs de u à partir de q_0^1 vers un état accepteur de F^1 et à partir de ces exécution et des transitions $(q^i, \epsilon, q_{\text{init}}^1)$ et celles de l'ensemble $\{(q_f^1, \epsilon, q_{\text{init}}^1) \mid q_f^1 \in F^1\}$, nous construisons une exécution de u sur A_* qui termine dans q^f . Ceci permet de conclure que u est accepté par A_* .

- La correction de cette transformation, repose sur la correction de la procédure de déterminisation (cf. propriété 5 (p. 169)) et celle du calcul de l'automate produit (cf. propriété 3 (p. 73)).

Solution de l'exercice 99 (page 178)

- Nous donnons quelques exemples de mots miroirs.
 - Considérons l'alphabet $\{a, b, c\}$.
 - Le mot *cba* est le miroir du mot *abc*.
 - Le mot *abb* est le mot miroir du mot *bba*.
 - Le mot *abba* est le mot miroir du mot *abba*.
 - Considérons l'alphabet latin (des mots du dictionnaire en français).
 - Le mot *son* est le mot miroir du mot *nos*.
 - Le mot *saper* est le mot miroir du mot *repas*.
 - Les mots *kayak*, *rever* et *monnom* sont les mots miroirs des mots *kayak*, *rever* et *monnom*, respectivement.

Les mots *abba*, *kayak*, *rever* et *monnom* sont des palindromes, leur miroirs sont égaux à eux mêmes. Les mots *son*, *nos*, *saper*, *repas* sont des anacycliques.

- Nous donnons quelques exemples de langages miroirs.
 - Le langage $\{cba, abb, abba\}$ est le langage miroir du langage $\{abc, bba, abba\}$.
 - Le langage $\{son, saper\}$ est le langage miroir du langage $\{nos, repas\}$.
 - Le langage $\{son, kayak, rever, monnom\}$ est le langage miroir de lui-même.
 - Soit Σ et Σ' deux alphabets tels que $\Sigma' \subseteq \Sigma$, le langage des mots qui commencent par un symbole de Σ' est le langage miroir des mots qui terminent par un symbole de Σ' .

Solution de l'exercice 100 (page 178)

- En suivant la définition des mots inductifs par la droite droite, voir définition 53 (p. 30), l'image miroir ou mot miroir d'un mot u , noté u^R , est défini inductivement par :
 - $\epsilon^R = \epsilon$,
 - $(u \cdot s)^R = s \cdot u^R$, pour tout $u \in \Sigma^*$, $s \in \Sigma$.

En suivant la définition des mots inductifs par la gauche, voir définition 54 (p. 30), l'image miroir ou mot miroir d'un mot u , noté u^R , est défini inductivement par :

- $\epsilon^R = \epsilon$,
- $(s \cdot u)^R = u^R \cdot s$, pour tout $u \in \Sigma^*$, $s \in \Sigma$.

Solution de l'exercice 101 (page 178)

1. Nous avons : $(w \cdot w')^R = w'^R \cdot w^R$.
2. Observons qu'en utilisant la définition de mot sous forme d'application, le résultat est quasi immédiat. Nous utilisons la définition des mots inductive à gauche, voir définition 54 (p. 30). Nous démontrons plutôt le résultat de la question précédente par induction sur $w \in \Sigma^*$.
 - Cas de base. Nous avons $\epsilon^R = \epsilon$ et $\epsilon \cdot w = w$.
 - Pas d'induction. Soit $w \in \Sigma^*$, supposons le résultat vérifié (hypothèse d'induction). Soit $s \in \Sigma$, démontrons que $((s \cdot w) \cdot w')^R = w'^R \cdot (s \cdot w)^R$.

Nous avons :

$$\begin{aligned}
 ((s \cdot w) \cdot w')^R &= (s \cdot (w \cdot w'))^R && (\text{associativité de la concaténation}) \\
 &= (w \cdot w')^R \cdot s && (\text{définition de l'opération miroir, exercice 100}) \\
 &= (w'^R \cdot w^R) \cdot s && (\text{hypothèse d'induction}) \\
 &= w'^R \cdot (w^R \cdot s) && (\text{associativité de la concaténation}) \\
 &= w'^R \cdot (s \cdot w)^R && (\text{définition de l'opération miroir, exercice 100})
 \end{aligned}$$

Solution de l'exercice 102 (page 178)

1. Soit L un langage à états reconnu par un AFED $A_L = (Q, \Sigma, q_{\text{init}}, \delta, F)$. Nous construisons l' ϵ -AFEND $R(A_L) = (Q \cup \{q_i\}, \Sigma, q_i, R(\delta) \cup \{(q_i, \epsilon, f) \mid f \in F\}, \{q_{\text{init}}\})$ avec $R(\delta) = \{(q', s, q) \mid (q, s, q') \in \delta\}$. $R(A_L)$ est obtenu à partir de A_L en ajoutant un nouvel état q_i qui devient l'état initial, le seul état accepteur est l'état initial de A_L , les transitions de A_L dans δ sont inversées, des transitions sont ajoutées entre le nouvel état initial q_i et chacun des états accepteurs de A_L .
Nous devons démontrer maintenant que si l'AFED A_L reconnaît le langage (à états) L , alors $R(A_L)$ reconnaît $R(L)$; démontrant ainsi que si L est un langage à l'état $R(L)$ l'est également. Pour cela, nous démontrons que si u est un mot accepté par A_L , alors $R(u)$ est accepté par $R(A_L)$.

Soit u un mot accepté par A_L . Soit $n \in \mathbb{N}$, la longueur de u . L'exécution de u sur A_L peut s'écrire $(q^0, u^0) \cdot (q^1, u^1) \cdots (q^n, u^n)$, où $u^n = \epsilon$, q^n est un état $q_f \in F$ et $q^0 = q_{\text{init}}$. Chaque dérivation depuis la configuration (q^i, u^i) vers la configuration (q^{i+1}, u^{i+1}) , avec $q^i, q^{i+1} \in Q$ et $u^i = u_{i+1} \dots$ est le suffixe à partir de la position $i+1$ de u (de longueur $n-i$) pour $i \in [0, n-1]$, est obtenue en suivant la fonction de transition δ de A_L . À chacune de ces dérivations, nous pouvons associer la dérivation depuis la configuration $(q^{n-(i+1)}, u^R_{i+1} \dots)$ vers la configuration $(q^{n-i}, u^R_{i+1} \dots)$, avec $u^R_{i+1} = u^R_{i+2} \dots$ le suffixe de u^R à partir de la position $i+1$, obtenue à partir de l'exécution de u^R sur $R(A_L)$. Cette propriété peut se démontrer facilement par récurrence sur la longueur de u . Ainsi, à l'exécution $(q_0, u) \cdots (q_f, \epsilon)$ du mot u sur A_L , nous pouvons associer la séquence de configurations $(q_f, u^R) \cdots (q_0, \epsilon)$ sur $R(A_L)$. Nous complétons cette séquence de configurations en une exécution de u^R sur $R(A_L)$ en utilisant une dérivation de configurations via la transition (q_i, ϵ, q_f) de $R(A_L)$: $(q_i, u^R) \cdot (q_f, u^R) \cdots (q_0, \epsilon)$. Comme q_0 est un état accepteur de $R(A_L)$, cela montre que u^R est accepté par $R(A_L)$.

Solution de l'exercice 103 (page 178)

1. Soit $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$, l'AFED reconnaissant L . Soit $h : \Sigma \rightarrow \Sigma^*$ le morphisme considéré. À chaque symbole de Σ , h associe un mot sur Σ' .

En réutilisant les résultats de l'exercice 83, à chaque transition (q, s, q') dans A sur le symbole s , on peut associer l' ϵ -AFEND suivant défini sur l'alphabet Σ' :

$$A_{q,h(s),q'} = (Q_{q,h(s),q} \cup \{q, q'\}, \Sigma', q, \Delta_{q,h(s),q'}, q')$$

qui reconnaît $h(s)$ avec l'état q comme état initial et q' comme (unique) état accepteur. Nous pouvons construire les automates $A_{q,h(s),q'}$ tels que les ensembles d'états $Q_{h(s)}$, pour $s \in \Sigma$, soient disjoints, c'est-à-dire $\forall s', s'' \in \Sigma. s' \neq s'' \implies Q_{h(s')} \cap Q_{h(s'')} = \emptyset$. Ceci est possible en utilisant au besoin un renommage des états.

Ainsi, l'automate reconnaissant $h(L)$ est l' ϵ -AFEND $(Q_h, \Sigma', q_{\text{init}}, \Delta_h, F)$ tel que :

- Q_h est l'ensemble d'états $\bigcup_{(q,s,q') \in \delta} Q_{q,s,q'}$, qui est l'union des ensembles des états des automates associés aux transitions de δ ;
- Σ' est l'alphabet;
- q_{init} est l'état initial, qui est l'état initial de A ;
- Δ_h est la relation de transition $\bigcup_{(q,s,q') \in \delta} \Delta_{q,h(s),q'}$ qui est l'union des ensembles de transitions des automates associés aux transitions de δ ;
- F est l'ensemble des états accepteurs (ensemble des états accepteurs de A).

Nous appelons $h(A)$ cet automate.

2. Pour la démonstration, pour une relation de transition Δ , nous définissons une fermeture transitive notée Δ^+ et définie comme suit :

- $\Delta^1 = \Delta$,
- $\Delta^{n+1} = \{(q, \alpha \cdot u, q') \mid \exists q'' (q, \alpha, q'') \in \Delta \wedge (q'', u, q') \in \Delta^n\}$, pour $n > 0$,
- $\Delta^+ = \bigcup_{n>0} \Delta^n$.

De plus, nous utilisons les deux propriétés intermédiaires suivantes :

1. $\forall w \in \Sigma^+. \forall q, q' \in Q. (q, w, q') \in \Delta^+ \iff (q, h(w), q') \in \Delta_h^+$,
2. $\forall q, q' \in Q. (q, x, q') \in \Delta_h^+ \implies \exists w \in \Sigma^*. h(w) = x$.

Démontrons la propriété 1 par induction sur $w \in \Sigma^+$.

- Cas de base. Le mot w est un symbole $s \in \Sigma$. La propriété découle directement des définitions de Δ^+ , Δ_h^+ et A_h , plus particulièrement des transitions et des états créés dans A_h à partir de (q, s, q') dans Δ .
- Pas d'induction. Soient $q, q' \in Q$, supposons $(q, s \cdot w, q') \in \Delta^+$. Il existe un état $q'' \in Q$ tel que $(q, s, q'') \in \Delta$ et $(q'', w, q') \in \Delta^+$. Par définition de A_h , de $(q, s, q'') \in \Delta$, nous déduisons $(q, h(s), q'') \in \Delta_h^+$. En utilisant l'hypothèse d'induction, de $(q'', w, q') \in \Delta^+$, nous déduisons $(q'', h(w), q') \in \Delta_h^+$. De $(q, h(s), q'') \in \Delta_h^+$ et $(q'', h(w), q') \in \Delta_h^+$, nous déduisons $(q, h(s) \cdot h(w), q') \in \Delta_h^+$, c'est-à-dire $(q, h(s \cdot w), q') \in \Delta_h^+$. Réciproquement, le même raisonnement peut être suivi dans le sens inverse pour obtenir le résultat attendu.

Démontrons la propriété 2 par récurrence forte sur la longueur de $x \in \Sigma^*$.

- Cas de base. Nous avons $x = \epsilon$. De $(q, \epsilon, q') \in \Delta_h^+$, nous déduisons $(q, \epsilon, q') \in \Delta_h$. Par construction de A_h , comme $q, q' \in Q$, ceci n'est possible que s'il existe $s \in \Sigma$ tel que $h(s) = \epsilon$.
- Pas de récurrence. Supposons $(q, x, q') \in \Delta_h^+$ avec $q, q' \in Q$ et x un mot de longueur n . Il existe $m \in \mathbb{N}$ tel que $(q, x_1, q_1) \in \Delta_h^+, (q_1, x_2, q_2) \in \Delta_h^+, \dots, (q_{m-1}, x_m, q') \in \Delta_h^+$. Notons que, d'après la définition de Δ_h^+ , nous avons $m \geq n$. Nous distinguons deux cas : Si $\forall i \in [1, m-1], q_i \notin Q$. Par définition de l'automate A_h , il existe $s \in \Sigma$ tel que $h(s) = x$. Si $\exists i \in [1, m-1], q_i \in Q$. Soit i_m le plus petit élément de $[1, m-1]$ tel que $q_{i_m} \in Q$. Le mot w s'écrit $w = x_1 \cdots x_{i_m} \cdot x_{i_m+1} \cdots x_{m-1}$. L'hypothèse de récurrence peut être appliquée sur $x_1 \cdots x_{i_m}$ et $x_{i_m+1} \cdots x_{m-1}$ pour déduire l'existence de deux mots w_1 et w_2 tels que $h(w_1) = x_1 \cdots x_{i_m}$ et $h(w_2) = x_{i_m+1} \cdots x_{m-1}$. Comme $h(w_1 \cdot w_2) = h(w_1) \cdot h(w_2)$, nous en déduisons le résultat attendu.
- Démontrons maintenant que $h(A)$ reconnaît $h(L)$. Observons que les automates A et $h(A)$ ont le même état initial et les mêmes états accepteurs.
 - $\mathcal{L}_{\text{auto}}(h(A)) \subseteq h(L)$. Considérons un mot $x \in \mathcal{L}_{\text{auto}}(h(A))$. Comme x est accepté par $h(A)$, il existe $q_f \in F$ tel que $(q_{\text{init}}, x, q_f) \in \Delta_h^+$. D'après la propriété 2, il existe $w \in \Sigma^*$ tel que $h(w) = x$. Ainsi, $(q_{\text{init}}, h(w), q_f) \in \Delta_h^+$. En utilisant la propriété 1, nous en déduisons que $(q_{\text{init}}, w, q_f) \in \Delta^+$. Ainsi, $w \in L$ et $h(w) = x \in h(L)$.
 - $\mathcal{L}_{\text{auto}}(h(A)) \supseteq h(L)$. Considérons un mot $x \in h(L)$. Il existe $w \in L$ tel que $h(w) = x$. Comme $w \in L$, il existe $q_f \in F$ tel que $(q_{\text{init}}, w, q_f) \in \Delta^+$. En utilisant la propriété 1, nous déduisons que $(q_{\text{init}}, x, q_f) \in \Delta_h^+$ et donc $x \in h(A)$.

Solution de l'exercice 104 (page 178)

1. Étant donnés un AFED $A_h = (Q_h, \Sigma_h, q_{\text{init}}, \delta_h, F)$ et un morphisme $h : \Sigma \rightarrow \Sigma_h^*$, on construit un AFEND $(Q, \Sigma, q_{\text{init}}, \Delta, F)$ tel que $h^{-1}(\mathcal{L}_{\text{auto}}(A_h)) = \mathcal{L}_{\text{auto}}(A)$ en deux étapes.
 1. pour tout $s \in \Sigma$ tel que $h(s) = \epsilon$, pour chaque $(q, h(s), q') \in \delta_h$, on ajoute q, q' dans Q et (q, s, q') dans Δ .
 2. pour tout $s \in \Sigma$ tel que $h(s) = \epsilon$, on ajoute (q, s, q') dans Δ pour tout $q \in Q$.
2. La preuve suit un principe similaire à celle de l'exercice 103. Nous utilisons la propriété suivante qui permet de montrer l'inclusion de $\mathcal{L}_{\text{auto}}(A)$ et $\mathcal{L}_{\text{auto}}(A_h)$ l'un dans l'autre : $\forall q, q' \in Q, (q, w, q') \in \Delta^+ \iff (q, h(w), q') \in \Delta_h^+$

8.5.5 Algorithmes sur les ϵ -AFEND

Solution de l'exercice 105 (page 179)

1. L'algorithme `epsilon_fermeture()` (algorithme 35) prend en paramètre un ϵ -AFEND et un de ses états q et retourne l' ϵ -fermeture de cet état. Cet algorithme est basé sur la définition inductive de l' ϵ -fermeture d'un état et calcule le point fixe de la

Algorithme 35 *epsilon_fermeture()* pour l' ϵ -fermeture d'un état

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ (* un ϵ -AFEND *)
Entrée : $q \in Q$ (* un état *)
Sortie : l' ϵ -fermeture de q (* un sous-ensemble de Q *)

```

1: ensemble d'états Fermeture := { $q$ } ; (* cas de base, déclaration et initialisation de Fermeture *)
2: ensemble d'états preFermeture :=  $\emptyset$  ; (* déclaration et initialisation de preFermeture, valeur à l'itération précédente *)
3: ensemble d'états tmp ; (* déclaration d'une variable temporaire *)
4: tant que Fermeture  $\neq$  preFermeture faire
5:   tmp := Fermeture \ preFermeture ; (* états utilisés pour trouver de nouveaux états dans l' $\epsilon$ -fermeture *)
6:   preFermeture := Fermeture ; (* sauvegarde de Fermeture *)
7:   Fermeture := Fermeture  $\cup$  { $q'' \in Q \mid \exists q' \in \text{tmp}. (q', \epsilon, q'') \in \Delta$ } ; (* états liés à un état de tmp par une  $\epsilon$ -transition *)
8: fin tant que
9: retourner Fermeture ;

```

fonction calculant l' ϵ -fermeture d'un ensemble d'états, appliquée initialement à l'état q . Pour cela, l'algorithme procède de manière itérative en maintenant un ensemble résultat Fermeture et un ensemble preFermeture qui contient la valeur de l'ensemble fermeture à l'itération précédente. L'ensemble preFermeture permet de déterminer lorsque le point fixe a été trouvé, c'est-à-dire lorsque les ensembles preFermeture et Fermeture sont égaux.

Solution de l'exercice 106 (page 179)

1. L'algorithme est le même que celui pour déterminiser les AFEND. Il faut simplement calculer l' ϵ -fermeture de l'état initial avant de l'ajouter aux états à visiter ainsi que l' ϵ -fermeture de chaque successeur q^{succ} après son calcul.

Solution de l'exercice 107 (page 179)

1. Il y a plusieurs solutions possibles pour résoudre le problème.
 La première utilise l'automate complémentaire, définition 79 (p. 72), et le test du langage vide, algorithme 11 (p. 100)). Nous construisons l'automate complémentaire de l'automate passé en paramètre. Pour cela, nous devons déterminiser l'automate. Nous pouvons ensuite calculer l'automate complémentaire. L'automate de départ reconnaît le langage universel si et seulement si l'automate complémentaire reconnaît le langage vide. Nous obtenons l'algorithme 36.
 La deuxième utilise la distinguabilité entre états, algorithme 30 (p. 123). Nous construisons un automate pour le langage universel sur l'alphabet de l'automate passé en paramètre. Ensuite, nous testons leur équivalence en utilisant l'algorithme de distinguabilité entre états. Pour cela, nous déterminisons puis complétons l'automate déterminisé. L'automate de départ reconnaît le langage universel si et seulement si les états initiaux de l'automate reconnaissant le langage universel et celui obtenu après déterminisation et complétion sont équivalents. Nous obtenons l'algorithme 37.

Algorithme 36 *langage_universel()* pour déterminer si un ϵ -AFEND reconnaît le langage universel

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ (* un ϵ -AFEND *)

Sortie : vrai si A reconnaît le langage universel sur Σ , faux sinon

- 1: $A_{\text{det}} = \text{determiniser}(A);$
 - 2: $A_{\text{det-neg}} = \text{complementer}(A_{\text{det}});$
 - 3: **retourner** *est_vide*($A_{\text{det-neg}}$);
-

Algorithme 37 *langage_universel()* pour déterminer si un ϵ -AFEND reconnaît le langage universel

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ (* un ϵ -AFEND *)

Sortie : vrai si A reconnaît le langage universel sur Σ , faux sinon

- 1: $A_{\text{det-comp-min}} = \text{minimiser}(\text{completer}(\text{determiniser}(A)))$
 - 2: $U = (\{q_0\}, \Sigma, q_0, \{(q_0, s, q_0) \mid s \in \Sigma\}, \{q_0\})$
(* U est l'automate minimal qui reconnaît le langage universel sur l'alphabet Σ . *)
 - 3: **retourner** *distinguable*($A_{\text{det-comp-min}}, U$);
-

La troisième utilise l'algorithme de minimisation, algorithme 31 (p. 124) et définition 103 (p. 125). Nous déterminons l'automate passé en paramètre, complétons le déterminisé puis le minimisons. Nous construisons un automate pour le langage universel sur l'alphabet de l'automate passé en paramètre. L'automate de départ reconnaît le langage universel si et seulement si l'automate reconnaissant le langage universel et l'automate obtenu après déterminisation/complétion/minimisation sont syntaxiquement les mêmes, modulo un renommage des états. Comme l'automate (minimal) reconnaissant le langage universel a une structure simple (un seul état et une transition depuis et vers cet état sur chaque symbole de l'alphabet), nous pouvons vérifier de manière simple que les deux automates ont la même structure en vérifiant que son ensemble d'états et son ensembles d'états finaux sont réduits à un seul état et ensuite vérifier que toutes les transitions sont depuis et vers ce même état. Notons que, comme l'automate est complet, nous savons qu'il y a une transition sur chaque symbole de l'alphabet depuis n'importe quel état. Nous obtenons l'algorithme 38.

2. Pour déterminer si un automate (sur un alphabet Σ) reconnaît le langage universel sur un alphabet Σ' , nous distinguons trois cas :

Le premier cas est celui où $\Sigma' \setminus \Sigma \neq \emptyset$, c'est-à-dire l'alphabet de l'automate ne contient pas tous les symboles de l'alphabet Σ' . Dans ce cas, nous pouvons conclure immédiatement que l'automate ne peut pas reconnaître le langage universel sur Σ' car il « manque » des symboles dans son alphabet. Le deuxième cas est celui où $\Sigma = \Sigma'$, c'est-à-dire l'alphabet de l'automate est le même que celui du langage universel. Dans ce cas, nous pouvons réutiliser l'algorithme de la question précédente en vérifiant que l'automate d'entrée reconnaît le langage universel (sur son alphabet).

Le troisième cas correspond aux autres situations. Nous commençons par vérifier qu'il n'y a pas de mot accepté impliquant un symbole de $\Sigma \setminus \Sigma'$. Pour cela, nous émondons l'automate (pour rendre tous les états accessibles et coaccessibles et vérifions l'absence

Algorithme 38 *langage_universel()* pour déterminer si un ϵ -AFEND reconnaît le langage universel

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ (* un ϵ -AFEND *)

Sortie : vrai si A reconnaît le langage universel sur Σ , faux sinon

- 1: $A_{\text{det-comp-min}} = \text{minimiser}(\text{completer}(\text{déterminiser}(A)))$
- 2: soit $(Q', \Sigma, q'_{\text{init}}, \delta', F')$ l'automate $A_{\text{det-comp-min}}$
(* $A_{\text{det-comp-min}}$ est complet car il résulte de l'opération de minimisation. *)
- 3: **si** $|Q'| = 1$ ou $|F'| = 1$ **alors**
- 4: **retourner** faux
- 5: **fin si**
- 6: **pour** $(q, s, q') \in \delta'$ **faire**
- 7: **si** $q \neq q'_{\text{init}}$ ou $q' \neq q'_{\text{init}}$ **alors**
- 8: **retourner** faux
- 9: **fin si**
- 10: **fin pour**
- 11: **retourner** vrai;

de transitions sur $\Sigma \setminus \Sigma'$. Ensuite, nous calculons le sous-automate de l'automate d'entrée restreint à l'alphabet Σ' (en restreignant sa fonction de transition) et vérifions que celui-ci reconnaît le langage universel. En suivant le raisonnement précédent, nous obtenons l'algorithme 39.

Algorithme 39 *langage_universel()* pour déterminer si un ϵ -AFEND reconnaît le langage universel sur un alphabet donné

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ (* un ϵ -AFEND *)

Entrée : Σ' (* un alphabet *)

Sortie : vrai si A reconnaît le langage universel sur Σ' , faux sinon

- 1: **si** $\Sigma' \setminus \Sigma \neq \emptyset$ **alors**
- 2: **retourner** faux;
- 3: **fin si**
- 4: **si** $\Sigma = \Sigma'$ **alors**
- 5: **retourner** *langage_universel*(A);
- 6: **fin si**
- 7: soit $(Q', \Sigma, q'_{\text{init}}, \delta', F') = \text{émonder}(\text{déterminiser}(A))$
- 8: **pour** $(q, s, q') \in \delta'$ **faire**
- 9: **si** $s \in \Sigma \setminus \Sigma'$ **alors**
- 10: **retourner** faux;
- 11: **fin si**
- 12: **fin pour**
- 13: **retourner** *langage_universel* $\left((Q', \Sigma, q'_{\text{init}}, \delta' \cap (Q' \times \Sigma' \times Q'), F')\right)$;

Expressions régulières

9.1 Résumé intuitif du chapitre

Nous abordons dans ce chapitre une autre façon de caractériser les langages à états en utilisant la notion d'**expression régulière**. Nous obtenons ainsi une *caractérisation inductive* des langages à états. En effet, étant donné un vocabulaire Σ , nous définissons comme base d'induction les symboles $, \epsilon, s \in \Sigma$ puis nous construisons l'**union** ensembliste avec l'opérateur $+$, la **concaténation** avec l'opérateur \cdot et l'**itération** avec l'opérateur $*$. Cette définition donne la **syntaxe des expressions régulières** dans le sens où elle permet de *construire* des expressions que nous appelons expressions régulières.

Nous donnons une **sémantique des expressions régulières** en termes de **langage** ; c'est-à-dire, à chaque expression régulière, nous associons un langage. Cette sémantique est définie par induction sur la structure des expressions régulières. Nous définissons la notion d'**équivalence** entre expressions régulières : deux expressions régulières sont équivalentes si et seulement si les langages associés par la sémantique sont égaux. Nous donnons des **identités remarquables** sur les expressions régulières et quelques méthodes pour la **simplification** d'expression régulière. Ainsi, les expressions régulières fournissent également une vision **algébrique** des langages et permettent le calcul.

Dans le chapitre suivant nous verrons comment associer un automate à une expression régulière et réciproquement comment associer une expression régulière à un automate.

D'un point de vue pratique, les expressions régulières sont utilisées :

- en compilation pour l'analyse lexicale lors par exemple de la description des mots clés d'un langage de programmation ;
- dans l'environnement des systèmes d'exploitation de la famille d'Unix pour les commandes `sed`, `awk`, `grep`, etc., qui permettent de rechercher et manipuler des chaînes de caractères (pour lesquels l'utilisation d'automates n'est pas envisageable) ;
- pour la spécification de nombreux formats, comme par exemple les emails valides dans les formulaires web.

9.2 Les notions essentielles

Dans la suite, nous considérons un alphabet Σ . Nous définissons les expressions régulières, leur syntaxe (section 9.2.1) et leur sémantique (section 9.2.2), quelques conventions de notations (section 9.2.3) et ensuite quelques unes de leur propriétés (section 9.2.5).

9.2.1 Syntaxe des expressions régulières

La syntaxe des expressions est définie inductivement.

Définition 127 (Syntaxe des expressions régulières) *L'ensemble des expressions régulières sur Σ , noté $ER(\Sigma)$ ou ER lorsque l'alphabet se déduit du contexte ou n'a pas d'importance, est défini inductivement comme suit :*

- la base $\{\epsilon, \emptyset\} \cup \Sigma$ définissant expressions régulières de base sur Σ ;
- les règles de construction (d'arité 2) dans $ER \times ER \rightarrow ER$ définies par $(e, e') \mapsto e \cdot e'$ et $(e, e') \mapsto e + e'$;
- la règle de construction (d'arité 1) dans $ER \rightarrow ER$ définie par $e \mapsto^*$.

La définition inductive précédente indique que :

- ϵ et \emptyset sont des expressions régulières sur Σ ;
 - si $s \in \Sigma$, alors s est une expression régulière sur Σ ;
 - si e et e' sont des expressions régulières sur Σ , alors $e \cdot e'$, $e + e'$ et e^* sont des expressions régulières sur Σ ; \cdot , $+$ et * sont des opérateurs de construction des expressions régulières.
- De plus, nous notons e^+ pour l'expression régulière $e \cdot e^*$.

9.2.2 Sémantique des expressions régulières

La sémantique des expressions régulières leur associe des langages.

Définition 128 (Sémantique des expressions régulières) *Le langage (unique) dénoté par une expression régulière est donné par l'application $\mathcal{L}_{ER} : ER(\Sigma) \rightarrow \mathcal{P}(\Sigma^*)$, définie inductivement, en suivant la syntaxe des expressions régulières; voir définition 127 :*

$$\begin{array}{ll} - \mathcal{L}_{ER}(\epsilon) = \{\epsilon\}, & - \mathcal{L}_{ER}(e + e') = \mathcal{L}_{ER}(e) \cup \mathcal{L}_{ER}(e'), \\ - \mathcal{L}_{ER}(\emptyset) = \emptyset, & - \mathcal{L}_{ER}(e \cdot e') = \mathcal{L}_{ER}(e) \cdot \mathcal{L}_{ER}(e'), \\ - \mathcal{L}_{ER}(s) = \{s\}, \text{ pour } s \in \Sigma, & - \mathcal{L}_{ER}(e^*) = \mathcal{L}_{ER}(e)^*. \end{array}$$

Remarque 38 *Les définitions 127 et 128, relatives à la syntaxe et la sémantique des expressions régulières, distinguent explicitement les opérateurs \cdot , $+$ et * qui sont des opérateurs syntaxiques s'appliquant aux expressions régulières, des opérateurs \cdot , \cup et $*$ qui sont des opérateurs sémantiques s'appliquant aux langages dénotés par les expressions régulières.*

L'expression régulière e^+ dénote le langage des mots qui sont formés par la concaténation d'au moins un mot dans le langage dénoté par l'expression régulière e .

Propriété 10 *Si $\epsilon \in \mathcal{L}_{ER}(e)$, alors $\mathcal{L}_{ER}(e^+) = \mathcal{L}_{ER}(e^*)$.*

Définition 129 (Langage régulier) *Un langage L est régulier si et seulement s'il existe une expression régulière e telle que $\mathcal{L}_{ER}(e) = L$.*

9.2.3 Conventions de notation et précédence des opérateurs

Nous utilisons les conventions de notations suivantes :

- *Opérateurs syntaxiques vs sémantiques.* Nous ne ferons plus la distinction explicitement entre les opérateurs syntaxiques et sémantiques et écrivons \cdot à la place de $_$ d'une part et $*$ à la place de $\underline{_}$ d'autre part.
- *Utilisation des parenthèses.* Nous nous permettons l'utilisation des parenthèses autant que nécessaire, entre autre pour faciliter la lecture des expressions régulières et réduire les ambiguïtés.
- *Utilisation de l'associativité.* Nous utilisons l'associativité gauche des opérateurs $+$ et \cdot provenant de l'associativité des opérateurs d'union ensembliste et de concaténation. Supposons que $\{a, b, c\} \subseteq \Sigma$, nous écrivons des expressions comme $a + b + c$ à la place de $(a + b) + c$ et $a + b^*$ à la place de $(a + (b^*))$.
- *Omission du symbole de concaténation.* Nous permettons l'omission du symbole de concaténation et écrivons ee' à la place de $e \cdot e'$, pour deux expressions régulières e et e' .

Pour éviter les ambiguïtés, en plus de l'utilisation des parenthèses, nous admettons les priorités suivantes pour l'application des opérateurs dans un ordre décroissant : $*$, \cdot , $+$. Ces priorités définissent la *précédence des opérateurs*.

9.2.4 Quelques propriétés : équivalence et simplification

Définition 130 (Équivalence entre deux expressions régulières) *Les expressions régulières e et e' sont dites équivalentes (resp. non équivalentes), noté $e \equiv e'$ (resp. $e \not\equiv e'$), lorsque $\mathcal{L}_{\text{ER}}(e) = \mathcal{L}_{\text{ER}}(e')$ (resp. $\mathcal{L}_{\text{ER}}(e) \neq \mathcal{L}_{\text{ER}}(e')$).*

Remarque 39 *La relation \equiv entre expressions régulières est effectivement une relation d'équivalence car la relation d'égalité est une relation d'équivalence sur les langages.*

Propriété 11 (Identités classiques entre expressions régulières) *Des équivalences entre expressions régulières correspondant aux identités dites classiques sont représentées dans le tableau 9.1 (p. 208). Démontrer ces identités fait l'objet de l'exercice 112 (p. 209).*

Propriété 12 (Principes de simplification d'expressions régulières) *Soient e_1 et e_2 deux expressions régulières. Nous utiliserons les principes suivants pour simplifier les expressions régulières :*

- *Si $e_1 \equiv e_2$, alors on peut substituer e_1 par e_2 dans une expression régulière sans changer le langage dénoté par cette expression.*
- *Si $\mathcal{L}_{\text{ER}}(e_1) \subseteq \mathcal{L}_{\text{ER}}(e_2)$, alors $\mathcal{L}_{\text{ER}}(e_1 + e_2) \equiv \mathcal{L}_{\text{ER}}(e_2)$. Donc, $e_1 + e_2$ peut être remplacée par e_2 dans une expression régulière sans changer le langage qu'elle dénote.*

9.2.5 Quelques problèmes de décision : équivalence et inclusion de langages dénotés

Définition 131 (Problème de l'équivalence entre expressions régulières) *Est-ce que deux expressions régulières sont équivalentes ?*

Tableau 9.1 – Identités classiques entre les expressions régulières. Dans le tableau, e , f et g sont des expressions régulières.

Expression régulière	Expression régulière équivalente	Remarque
$e + \emptyset$	e	
$e \cdot \epsilon$	e	trivial
$e \cdot \emptyset$	\emptyset	
$(e + f) + g$	$e + (f + g)$	associativité
$(e \cdot f) \cdot g$	$e \cdot (f \cdot g)$	
$e \cdot (f + g)$	$(e \cdot f) + (e \cdot g)$	distributivité (de la concaténation sur l'union)
$(e + f) \cdot g$	$(e \cdot g) + (f \cdot g)$	
$e + f$	$f + e$	commutativité
e^*	$\epsilon + e \cdot e^*$	apéridicité
e^*	$\epsilon + e^* \cdot e$	
$(\emptyset)^*$	ϵ	définition de la fermeture de Kleene
$e + e$	e	
$(e^*)^*$	e^*	idempotence

Définition 132 (Problème de l'inclusion des langages dénotés par des expressions régulières)
Est-ce que le langage dénoté par une expression régulière est inclus dans le langage dénoté par une autre expression régulière ?

9.3 Exercices

Exercice 108 (♠) — Trouver des expressions régulières

Considérons l'alphabet $\Sigma = \{a, b, c\}$. Pour chacun des langages suivants sur Σ , donner une expression régulière qui le dénote.

1. L'ensemble des mots qui commencent par a et finissent par b .
2. L'ensemble des mots qui contiennent au moins trois occurrences du symbole b .
3. L'ensemble des mots qui contiennent au moins trois occurrences *consécutives* du symbole b .

4. L'ensemble des mots qui contiennent au moins trois symboles et le troisième symbole est a .

Exercice 109 (♠♦) — Trouver des expressions régulières

Considérons l'alphabet $\Sigma = \{a, b, c\}$. Pour chacun des langages suivants sur Σ , donner une expression régulière qui le dénote.

1. L'ensemble des mots qui contiennent un nombre pair de a .
2. L'ensemble des mots qui contiennent un nombre impair de a .
3. L'ensemble des mots qui contiennent un nombre de a multiple de 3.
4. L'ensemble des mots de longueur impaire.
5. L'ensemble des mots de longueur au moins 1 et au plus 3.

Exercice 110 (♠♦♦) — Trouver des expressions régulières

Considérons l'alphabet $\Sigma = \{a, b, c\}$. Pour chacun des langages suivants sur Σ , donner une expression régulière qui le dénote.

1. L'ensemble des mots qui ne contiennent pas le facteur $a \cdot a$.
2. L'ensemble des mots qui ne contiennent pas le facteur $a \cdot a \cdot b$.
3. L'ensemble des mots qui contiennent au moins 2 occurrences du symbole a , mais non consécutives.
4. L'ensemble des mots qui commencent et finissent par le même symbole.

Exercice 111 (♠♦) — Combien de mots d'une longueur donnée dans le langage

Considérons l'alphabet $\Sigma = \{a, b\}$. Pour chacune des expressions régulières suivantes, dire combien de mots de longueur 100 sont dans le langage qu'elles décrivent.

1. $a \cdot (a + b)^* \cdot b$.
2. $a^* \cdot b \cdot a \cdot b^*$.
3. $(a + b \cdot a)^*$.

Exercice 112 (♠♦♦) — Preuves des identités classiques

Nous considérons les identités classiques entre expressions régulières dans le tableau 9.1 (p. 208).

1. Démontrer chacune de ces identités.

Exercice 113 (♠♦♦) — Équivalence ou non entre expressions régulières

Soient R et S des expressions régulières. Donner une preuve ou un contre-exemple pour les équivalences candidates suivantes entre expressions régulières.

- | | |
|--------------------------------------------|----------------------------------------------|
| 1. $(\epsilon + R)^* \equiv R^*$. | 5. $(RS + R)^* RS \equiv (RR^* S)^*$. |
| 2. $(\epsilon + R) \cdot R^* \equiv R^*$. | 6. $(R + S)^* S \equiv (R^* S)^*$. |
| 3. $(R + S)^* \equiv R^* + S^*$. | |
| 4. $(RS + R)^* R \equiv R(SR + R)^*$. | 7. $S(RS + S)^* R \equiv RR^* S(RR^* S)^*$. |

Exercice 114 (♠♦) — Taille d'une expression régulière et nombre d'états

Nous considérons l'alphabet $\Sigma = \{a, b\}$. Nous nous intéressons à la relation entre la taille d'une expression régulière, le nombre d'états de l'AFEND et de l'AFED correspondant à cette expression.

1. Considérons l'expression régulière $E_n = \Sigma^* \cdot \Sigma^n$. Donner la forme générale de l'AFEND reconnaissant E_n .
2. Donner le nombre d'états des AFED reconnaissant E_1, E_2 et E_3 .
3. Extrapoler le nombre d'états de l'AFED qui reconnaît E_n .
4. Reprendre les questions précédentes avec l'expression régulière $E'_n = \Sigma^* \cdot b \cdot \Sigma^n$.
5. Comparer les tailles des AFEND et AFED pour E_n et E'_n .

Exercice 115 (♠♦) — Simplifier des expressions régulières

Considérons l'alphabet $\Sigma = \{a, b\}$. Simplifier chacune des expressions régulières suivantes, c'est-à-dire trouver une expression régulière équivalente qui contienne moins de symboles.

1. $\epsilon + a \cdot b + a \cdot b \cdot a \cdot b \cdot (a \cdot b)^*$.
2. $a \cdot a \cdot (b^* + a) + a \cdot (a \cdot b^* + a \cdot a)$.
3. $a \cdot (a + b)^* + a \cdot a \cdot (a + b)^* + a \cdot a \cdot a \cdot (a + b)^*$.

Exercice 116 (♠♦♦) — Fermeture par opération miroir

Nous souhaitons démontrer que l'ensemble des expressions régulières est fermé par l'opération miroir.

1. Définir une fonction miroir : $ER \rightarrow ER$ qui prend en entrée une expression régulière et produit en sortie une expression régulière telle que le langage dénoté par l'expression régulière de sortie soit le miroir du langage dénoté par l'expression régulière d'entrée.
2. Démontrer que la fonction trouvée à la question précédente est correcte. C'est-à-dire, démontrer que :

$$\forall e \in ER, \mathcal{L}_{ER}(\text{miroir}(e)) = (\mathcal{L}_{ER}(e))^R.$$

9.4 Indications pour résoudre les exercices

Indications pour l'exercice 110 (p. 209)

1. La difficulté pour cette expression régulière est que l'expression régulière recherchée décrit la forme des mots qui ne possèdent pas un certain facteur. De tels mots ont trois parties : une pour le début, milieu et fin.
2. Le principe est le même qu'à la question précédente.
3. Identifier éventuellement deux interprétations de l'ensemble des mots.

Indications pour l'exercice 111 (p. 209)

1. Identifier les positions pour lesquelles un choix de symbole est possible. Dénombrer les choix.
2. Remarquer le nombre d'occurrences du symbole a en début de mot détermine le nombre d'occurrences du symbole b en fin de mot.
3. Définir la suite arithmético-géométrique des mots de longueur n dans ce langage.

Indications pour l'exercice 112 (p. 209)

Montrer l'égalité des langages dénotés par les expressions régulières.

Indications pour l'exercice 113 (p. 209)

Lorsque l'équivalence est fausse, considérer des langages réduits à un mot pour trouver le contre-exemple. Lorsque l'équivalence est vraie, montrer l'inclusion des langages dénotés l'un dans l'autre.

Indications pour l'exercice 115 (p. 210)

Pour chaque expression régulière, identifier des parties qui dénotent des langages dénotés par d'autres parties.

Indications pour l'exercice 116 (p. 210)

1. Définir une fonction qui prend une expression régulière en entrée et produit en sortie une expression régulière telle que le langage dénoté par l'expression régulière produite soit le langage miroir du langage dénoté par l'expression régulière d'entrée. Pour cela, définir la fonction par induction sur les expressions régulières, en suivant les cas de la définition des expressions régulières, voir définition 127 (p. 206).
2. Démontrer le résultat par induction sur l'expression régulière e .

9.5 Solutions des exercices

Solution de l'exercice 108 (page 208)

Pour chacun des langages proposés, nous donnons l'expression régulière correspondante. Les expressions régulières suivantes utilisent la précédence des opérateurs entre expressions régulières.

1. L'ensemble des mots qui commencent par a et finissent par b est dénoté par l'expression régulière : $a \cdot \Sigma^* \cdot b$.
2. L'ensemble des mots qui contiennent au moins trois occurrences du symbole b est dénoté par l'expression régulière : $(a + c)^* \cdot b \cdot (a + c)^* \cdot b \cdot (a + c)^* \cdot b \cdot \Sigma^*$.
3. L'ensemble des mots qui contiennent au moins trois occurrences consécutives du symbole b est dénoté par l'expression régulière : $\Sigma^* \cdot b \cdot b \cdot b \cdot \Sigma^*$.

4. L'ensemble des mots qui contiennent au moins 3 symboles et le troisième symbole est a est dénoté par l'expression régulière : $\Sigma \cdot \Sigma \cdot a \cdot \Sigma^*$.

Solution de l'exercice 109 (page 209)

Pour chacun des langages proposés, nous donnons l'expression régulière correspondante. Les expressions régulières utilisent la précédence des opérateurs entre expressions régulières.

1. L'ensemble des mots qui contiennent un nombre pair de a est dénoté par l'expression régulière : $(a \cdot (b + c)^* \cdot a + b + c)^*$.
2. L'ensemble des mots qui contiennent un nombre impair de a est dénoté par l'expression régulière : $(a \cdot (b + c)^* \cdot a + b + c)^* \cdot a \cdot (b + c)^*$.
3. L'ensemble des mots qui contiennent un nombre de a multiple de 3 est dénoté par l'expression régulière : $((b + c)^* \cdot a \cdot (b + c)^* \cdot a \cdot (b + c)^* \cdot a)^* \cdot (b + c)^*$.
4. L'ensemble des mots de longueur impaire est dénoté par l'expression régulière : $(\Sigma \cdot \Sigma)^* \cdot \Sigma$.
5. L'ensemble des mots de longueur au moins 1 et au plus 3 est dénoté par l'expression régulière : $\Sigma \cdot (\epsilon + \Sigma + \Sigma \cdot \Sigma)$.

Solution de l'exercice 110 (page 209)

Pour chacun des langages proposés, nous donnons l'expression régulière correspondante. Les expressions régulières utilisent la précédence des opérateurs entre expressions régulières, voir section 9.2.3 (p. 207).

1. L'ensemble des mots qui ne contiennent pas le facteur $a \cdot a$ est dénoté par l'expression régulière : $(b + c)^* \cdot (a \cdot (b + c)^+)^* \cdot (a + \epsilon)$.
2. L'ensemble des mots qui ne contiennent pas le facteur $a \cdot a \cdot b$ est dénoté par l'expression régulière : $(b + c + a \cdot (b + a^* \cdot c))^* \cdot a^*$.
3. Notons qu'il est possible d'interpréter la description du langage de deux manières.
 - L'ensemble des mots qui contiennent au moins 2 occurrences non consécutives du symbole a et où les deux premières occurrences du symbole a doivent être non consécutives (et les autres non contraintes) est dénoté par l'expression régulière : $(b + c)^* \cdot a \cdot (b + c)^+ \cdot a \cdot \Sigma^*$.
 - L'ensemble des mots qui contiennent au moins 2 occurrences non consécutives du symbole a est dénoté par l'expression régulière : $\Sigma^* \cdot a \cdot (b + c)^+ \cdot a \cdot \Sigma^*$.
4. L'ensemble des mots qui commencent et finissent par le même symbole est dénoté par l'expression régulière : $a \cdot \Sigma^* \cdot a + b \cdot \Sigma^* \cdot b + c \cdot \Sigma^* \cdot c + a + b + c + \epsilon$.

Solution de l'exercice 111 (page 209)

1. Pour former un mot de longueur 100, pour chacun des symboles du mot sauf le premier et le dernier, nous avons le choix entre le symbole a ou le symbole b . Ainsi, nous avons 98 positions dans le mot pour lequel nous avons deux choix possibles. Chacun de ces choix est indépendant. Ainsi, nous obtenons 2^{98} mots différents de longueur 100 pour ce langage.
2. Pour former un mot de longueur 100, nous avons le choix pour les symboles du début du mot et de la fin du mot. Nous pouvons placer des a en début de mot et

des symboles b en fin du mot. Nous notons que, comme la longueur du mot est fixé, le nombre d'occurrences du symbole a en début de mot détermine (exactement) le nombre d'occurrences du symbole b en fin de mot. Dans l'éventail des possibilités, nous pouvons choisir de mettre entre 0 et 98 occurrences du symbole a . Ainsi, nous obtenons 99 mots différents de longueur 100 pour ce langage.

3. Les mots de ce langage sont formés par la concaténation de facteurs qui sont soit a soit $b \cdot a$. En formant un mot de longueur 100 dans l'ordre des préfixes croissants, nous faisons un choix qui réduit la longueur du suffixe pour lequel nous avons le choix, soit de 1 symbole soit de 2 symboles. Nous définissons $(u_n)_{n \in \mathbb{N}}$ la suite telle que, pour $n \in \mathbb{N}$, u_n est le nombre de mots de longueur n dans ce langage. En se basant sur le raisonnement précédent, nous pouvons définir u_n par $u_0 = 1$, $u_1 = 1$ et $u_n = u_{n-1} + u_{n-2}$ pour $n > 1$. Ainsi, le nombre de mots de longueur 100 est le terme u_{100} de la suite qui peut être calculé en utilisant la relation de récurrence de la suite. Nous pouvons donner une expression directe du terme u_{100} : $u_{100} = \alpha\varphi^{100} + \beta\varphi'^{100}$ avec $\alpha = \frac{1}{2} + \frac{1}{2\sqrt{5}}$, $\beta = \frac{1}{2} - \frac{1}{2\sqrt{5}}$, $\varphi = \frac{1+\sqrt{5}}{2}$ et $\varphi' = \frac{1-\sqrt{5}}{2}$.

Solution de l'exercice 112 (page 209)

1. Nous considérons les équivalences tour à tour. Soient e, f, g trois expressions régulières sur un alphabet Σ . Les observations suivantes utilisent directement la sémantique des expressions régulières ; voir définition 128 (p. 206).

- Nous avons $e + \emptyset \equiv e$ car $\mathcal{L}_{\text{ER}}(e + \emptyset) = \mathcal{L}_{\text{ER}}(e) \cup \mathcal{L}_{\text{ER}}(\emptyset) = \mathcal{L}_{\text{ER}}(e) \cup \emptyset = \mathcal{L}_{\text{ER}}(e)$ car \emptyset est l'élément neutre de l'opérateur \cup , d'union ensembliste.
- Nous avons $e \cdot \epsilon \equiv e$ car $\mathcal{L}_{\text{ER}}(e \cdot \epsilon) = \mathcal{L}_{\text{ER}}(e) \cdot \mathcal{L}_{\text{ER}}(\epsilon) = \mathcal{L}_{\text{ER}}(e) \cdot \{\epsilon\} = \{w \cdot \epsilon \mid w \in \mathcal{L}_{\text{ER}}(e)\} = \{w \mid w \in \mathcal{L}_{\text{ER}}(e)\} = \mathcal{L}_{\text{ER}}(e)$, par définition de l'opérateur \cdot , de concaténation ensembliste, et la sémantique de la concaténation.
- Nous avons $e \cdot \emptyset \equiv e$ car $\mathcal{L}_{\text{ER}}(e \cdot \emptyset) = \mathcal{L}_{\text{ER}}(e) \cdot \mathcal{L}_{\text{ER}}(\emptyset) = \mathcal{L}_{\text{ER}}(e) \cdot \emptyset = \emptyset$, par définition de l'opérateur de concaténation que l'on applique en particulier à l'ensemble vide.
- Nous avons $(e + f) + g \equiv e + (f + g)$ car $\mathcal{L}_{\text{ER}}((e + f) + g) = \mathcal{L}_{\text{ER}}(e + f) \cup \mathcal{L}_{\text{ER}}(g) = (\mathcal{L}_{\text{ER}}(e) \cup \mathcal{L}_{\text{ER}}(f)) \cup \mathcal{L}_{\text{ER}}(g) = \mathcal{L}_{\text{ER}}(e) \cup (\mathcal{L}_{\text{ER}}(f) \cup \mathcal{L}_{\text{ER}}(g)) = \mathcal{L}_{\text{ER}}(e + (f + g))$, par associativité de l'opérateur d'union ensembliste.
- Nous avons $(e \cdot f) \cdot g \equiv e \cdot (f \cdot g)$, de manière similaire par associativité de l'opérateur de concaténation sur les langages.
- Nous avons $e \cdot (f + g) \equiv (e \cdot f) + (e \cdot g)$ car :

$$\begin{aligned}
 \mathcal{L}_{\text{ER}}(e \cdot (f + g)) &= \mathcal{L}_{\text{ER}}(e) \cdot \mathcal{L}_{\text{ER}}(f + g) \\
 &= \mathcal{L}_{\text{ER}}(e) \cdot (\mathcal{L}_{\text{ER}}(f) \cup \mathcal{L}_{\text{ER}}(g)) \\
 &= \{w_e \mid w_e \in \mathcal{L}_{\text{ER}}(e)\} \\
 &= \{w_e \cdot w \mid w_e \in \mathcal{L}_{\text{ER}}(e) \wedge (w \in \mathcal{L}_{\text{ER}}(f) \vee w \in \mathcal{L}_{\text{ER}}(g))\} \\
 &= \{w_e \cdot w \mid (w_e \in \mathcal{L}_{\text{ER}}(e) \wedge w \in \mathcal{L}_{\text{ER}}(f)) \\
 &\quad \vee (w_e \in \mathcal{L}_{\text{ER}}(e) \wedge w \in \mathcal{L}_{\text{ER}}(g))\} \\
 &= \{w_e \cdot w \mid w_e \in \mathcal{L}_{\text{ER}}(e) \wedge w \in \mathcal{L}_{\text{ER}}(f)\} \\
 &\quad \cup \{w_e \cdot w \mid w_e \in \mathcal{L}_{\text{ER}}(e) \wedge w \in \mathcal{L}_{\text{ER}}(g)\}.
 \end{aligned}$$

- Nous avons $(e + f) \cdot g \equiv (e \cdot g) + (f \cdot g)$, en suivant un principe similaire à la question précédente.
- Nous avons $e + f \equiv f + e$, en utilisant la commutativité de l'opérateur d'union ensembliste.
- Nous avons $e^* \equiv \epsilon + e \cdot e^* \equiv \epsilon + e^* \cdot e$ car $\mathcal{L}_{\text{ER}}(e^*) = \bigcup_{i \in \mathbb{N}} (\odot^i \mathcal{L}_{\text{ER}}(e))$ où $\odot^i \mathcal{L}_{\text{ER}}(e)$ est l'ensemble des mots formés par i concaténations de mots de $\mathcal{L}_{\text{ER}}(e)$ et ϵ est le mot formé par 0 concaténation de mots de $\mathcal{L}_{\text{ER}}(e)$.
- Nous avons $(\emptyset)^* = \{\epsilon\}$ car la fermeture de Kleene d'un langage contient le mot ϵ et les mots formés par concaténation de mots de ce langage, qui est ici l'ensemble vide.
- Nous avons $e + e \equiv e$ car $\mathcal{L}_{\text{ER}}(e + e) = \mathcal{L}_{\text{ER}}(e) \cup \mathcal{L}_{\text{ER}}(e) = \mathcal{L}_{\text{ER}}(e)$.
- Nous avons $(e^*)^* \equiv e^*$ car $\mathcal{L}_{\text{ER}}((e^*)^*) = \mathcal{L}_{\text{ER}}(e^*)$ ce que nous pouvons montrer en montrant l'inclusion des ensembles l'un dans l'autre. Considérons un mot de $\mathcal{L}_{\text{ER}}((e^*)^*)$, il peut s'écrire $e_1 \cdots e_n$, avec $n \in \mathbb{N}$ et $e_i \in \mathcal{L}_{\text{ER}}(e^*)$, $i \in [1, n]$. Chaque mot $e_i \in \mathcal{L}_{\text{ER}}(e^*)$ peut s'écrire $e_i^1 \cdots e_i^{n_i}$ avec $n_i \in \mathbb{N}$ et $e_i^j \in \mathcal{L}_{\text{ER}}(e)$ pour $j \in [1, n_i]$. Ainsi, chaque mot e_i est une concaténation de mots de $\mathcal{L}_{\text{ER}}(e)$ et donc $e_1 \cdots e_n$ est une concaténation de mots de $\mathcal{L}_{\text{ER}}(e)$. Ainsi, $\mathcal{L}_{\text{ER}}((e^*)^*) \subseteq \mathcal{L}_{\text{ER}}(e^*)$.

Solution de l'exercice 113 (page 209)

Lorsque l'équivalence entre deux expressions régulières est vraie, nous montrons l'inclusion des langages dénotés par les expressions régulières l'un dans l'autre.

1. Cette équivalence est vraie : $(\epsilon + R)^* \equiv R^*$.
 - $\mathcal{L}_{\text{ER}}((R + \epsilon)^*) \subseteq \mathcal{L}_{\text{ER}}(R^*)$. Soit $w \in \mathcal{L}_{\text{ER}}((R + \epsilon)^*)$. D'après la sémantique des expressions régulières, soit i) w est ϵ soit ii) s'écrit $w_1 \cdot w_2 \cdots w_n$ avec $w_i \in \mathcal{L}_{\text{ER}}(R + \epsilon)$. Nous distinguons deux cas. Le premier cas est celui où $w = \epsilon$. Dans ce cas $w \in \mathcal{L}_{\text{ER}}(R^*)$ d'après la sémantique de R^* (fermeture de Kleene de $\mathcal{L}_{\text{ER}}(R)$). Le deuxième cas est celui où w est formé par la concaténation de mots de $\mathcal{L}_{\text{ER}}(R)$ et ϵ . Ce mot peut donc s'écrire $w = w_1 \cdots w_m$ avec $m \leq n$ et $w_i = w'_i$ pour $1 \leq i \leq m$. Donc $w \in (\mathcal{L}_{\text{ER}}(R))^* = \mathcal{L}_{\text{ER}}(R^*)$.
 - $\mathcal{L}_{\text{ER}}((R + \epsilon)^*) \supseteq \mathcal{L}_{\text{ER}}(R^*)$. On a $\mathcal{L}_{\text{ER}}(R^*) = (\mathcal{L}_{\text{ER}}(R))^* \subseteq (\mathcal{L}_{\text{ER}}(R) \cup X)^*$, pour n'importe quel langage X et en particulier lorsque $X = \{\epsilon\}$.
2. Cette équivalence est vraie : $(\epsilon + R) \cdot R^* \equiv R^*$. En utilisant la distributivité de la concaténation sur l'union, nous avons $(\epsilon + R) \cdot R^* = \epsilon \cdot R^* + R \cdot R^*$. En utilisant $\mathcal{L}_{\text{ER}}(R \cdot R^*) \subseteq \mathcal{L}_{\text{ER}}(R^*)$, nous avons $R^* + R \cdot R^* = R^*$.
3. Cette équivalence est fausse : $(R + S)^* \neq R^* + S^*$. Comme contre-exemple, prenons $\Sigma = \{r, s\}$, $R = \{r\}$, $S = \{s\}$. Alors, $r \cdot s \in \mathcal{L}_{\text{ER}}((R + S)^*)$, mais $r \cdot s \notin \mathcal{L}_{\text{ER}}(R^* + S^*)$.
4. Cette équivalence est vraie : $(RS + R)^* R \equiv R(SR + R)^*$.
 - Preuve de $\mathcal{L}_{\text{ER}}((RS + R)^* R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$. En utilisant la définition de la fermeture de Kleene, nous devons démontrer que, pour tout $i \geq 0$, $\bigcup_{i \in \mathbb{N}} \mathcal{L}_{\text{ER}}((RS + R)^i R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^i)$, ce que nous pouvons démontrer par récurrence sur i .

- Pour $i = 0$ nous avons $\bigcup_{i=0} \mathcal{L}_{\text{ER}}((RS + R)^i R) = \mathcal{L}_{\text{ER}}(R)$ et $\mathcal{L}_{\text{ER}}(R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$.
- Considérons $i \in \mathbb{N}$ et supposons que la propriété soit vérifiée. Démontrons que $\mathcal{L}_{\text{ER}}((RS + R)^{i+1} R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$. Soit $w \in \mathcal{L}_{\text{ER}}((RS + R)^{i+1} R)$, nous avons $w = u_1 \dots u_i u_{i+1} r$, avec $\forall k \in [1, i+1], u_k \in \mathcal{L}_{\text{ER}}(RS + R)$, $r \in \mathcal{L}_{\text{ER}}(R)$, et $w' = u_1 \dots u_{i+1} r \in \mathcal{L}_{\text{ER}}((RS + R)^i R)$. Alors, $w = u_1 w'$ avec $u_1 \in \mathcal{L}_{\text{ER}}(RS + R)$ et $w' \in \mathcal{L}_{\text{ER}}((RS + R)^i R)$. Nous distinguons deux cas.
- Cas $u_1 \in \mathcal{L}_{\text{ER}}(R)$. Nous avons $w \in \mathcal{L}_{\text{ER}}(RR(SR + R)^i)$ et $w \in \mathcal{L}_{\text{ER}}(R(SR + R)^{i+1})$ car $\mathcal{L}_{\text{ER}}(R) \subseteq \mathcal{L}_{\text{ER}}(SR + R)$.
- Cas $u_1 \in \mathcal{L}_{\text{ER}}(RS)$. Nous avons $w \in \mathcal{L}_{\text{ER}}(RSR(SR + R)^i)$, et $w \in \mathcal{L}_{\text{ER}}(R(SR + R)^{i+1})$ car $\mathcal{L}_{\text{ER}}(SR) \subseteq \mathcal{L}_{\text{ER}}(SR + R)$.

Dans tous les cas, $w \in \mathcal{L}_{\text{ER}}(R(SR + R)^{i+1}) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$.

Finalement, nous avons $\bigcup_{k \in [1, i]} \mathcal{L}_{\text{ER}}((RS + R)^k R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$ et $\mathcal{L}_{\text{ER}}((RS + R)^{i+1} R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$. En conséquence, nous obtenons $\bigcup_{k \in [1, i+1]} \mathcal{L}_{\text{ER}}((RS + R)^k R) \subseteq \mathcal{L}_{\text{ER}}(R(SR + R)^*)$.

- Preuve de $\mathcal{L}_{\text{ER}}(R(SR + R)^*) \subseteq \mathcal{L}_{\text{ER}}((RS + R)^* R)$. Nous pouvons utiliser le même principe, à savoir de démontrer $\mathcal{L}_{\text{ER}}(R) \cup \bigcup_{i \in \mathbb{N}} \mathcal{L}_{\text{ER}}((SR + R)^i) \subseteq \mathcal{L}_{\text{ER}}((RS + R)^* R)$, par récurrence sur $i \in \mathbb{N}$.
5. Cette équivalence est fausse : $(RS + R)^* RS \not\equiv (RR^* S)^*$. Comme contre-exemple, prenons $R = \{r\}, S = \{s\}$. Alors, $\epsilon \notin \mathcal{L}_{\text{ER}}((RS + R)^* RS)$, mais $\epsilon \in \mathcal{L}_{\text{ER}}((RR^* S)^*)$.
 6. Cette équivalence est fausse : $(R + S)^* S \not\equiv (R^* S)^*$. Comme contre-exemple, prenons $S = \{s\}$. Alors, $\epsilon \notin \mathcal{L}_{\text{ER}}((R + S)^* S)$, mais $\epsilon \in \mathcal{L}_{\text{ER}}((R^* S)^*)$.
 7. Cette équivalence est fausse : $S(RS + S)^* R \not\equiv RR^* S(RR^* S)^*$. Comme contre-exemple, prenons $S = \{s\}, R = \{r\}$. Alors, $s \cdot r \in \mathcal{L}_{\text{ER}}(SR) \subseteq \mathcal{L}_{\text{ER}}(S(RS + S)^* R)$, mais $s \cdot r \notin \mathcal{L}_{\text{ER}}(RR^* S(RR^* S)^*)$.

Solution de l'exercice 114 (page 210)

1. La forme générale de l'AFEND reconnaissant le langage dénoté par E_n est donnée dans la figure 9.1a.
2. Partant des AFEND reconnaissant E_1, E_2 et E_3 et en les déterminisant, nous obtenons les AFED représentés dans les Figures 9.1b, 9.1c et 9.1d, respectivement. Ainsi, nous trouvons que les AFED correspondant possèdent, respectivement 2, 3 et 4 états.
3. Nous extrapolons que l'AFED reconnaissant E_n possède n états.
4. La forme générale de l'automate reconnaissant le langage dénoté par E'_n est donnée dans la figure 9.1e. Cet automate possède $n + 2$ états. Nous déterminisons les AFEND reconnaissant les langages dénotés par E'_1, E'_2 et E'_3 . Ces AFED sont ceux trouvés dans l'exercice 78 (p. 150) et sont représentés dans les Figures 7.13a, 7.13b et 7.13c, respectivement. Nous trouvons que les AFED correspondant possèdent respectivement 2, 4 et 8 états. Nous extrapolons que l'AFED reconnaissant E'_n possède 2^n états.

5. Un AFED reconnaissant le langage dénoté par E_n possède $\mathcal{O}(n)$ états, alors qu'un AFED pour E'_n possède $\mathcal{O}(2^n)$ états. Bien que les expressions régulières soient sensiblement de la même taille, elles correspondent à des AFED avec des tailles qui diffèrent de l'ordre d'une exponentielle. Un AFEND pour E'_n est l'exemple typique d'AFEND avec lequel la déterminisation « se passe mal ».

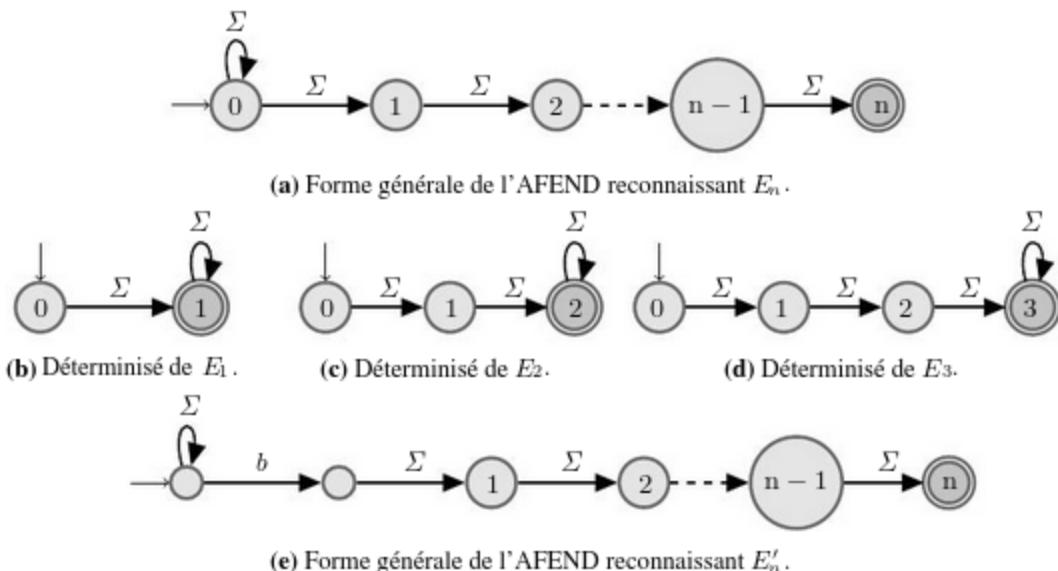


Figure 9.1 – Automates obtenus en solution de l'exercice 114.

Solution de l'exercice 115 (page 210)

Les expressions régulières simplifiées sont données ci-dessous.

1. Nous observons que $\mathcal{L}_{\text{ER}}(\epsilon) \subseteq \mathcal{L}_{\text{ER}}((a \cdot b)^*)$, $\mathcal{L}_{\text{ER}}(a \cdot b) \subseteq \mathcal{L}_{\text{ER}}((a \cdot b)^*)$ et $\mathcal{L}_{\text{ER}}(a \cdot b \cdot a \cdot b) \subseteq \mathcal{L}_{\text{ER}}((a \cdot b)^*)$. L'expression régulière simplifiée est : $(a \cdot b)^*$.
2. Nous avons $a \cdot a \cdot (b^* + a) \equiv a \cdot (a \cdot b^* + a \cdot a)$. L'expression régulière simplifiée est : $a \cdot a \cdot (b^* + a)$.
3. Nous avons $\mathcal{L}_{\text{ER}}(a \cdot a \cdot a(a + b)^*) \subseteq \mathcal{L}_{\text{ER}}(a \cdot a \cdot (a + b)^*) \subseteq \mathcal{L}_{\text{ER}}(a \cdot (a + b)^*)$. L'expression régulière simplifiée est : $a \cdot (a + b)^*$.

Solution de l'exercice 116 (page 210)

1. Nous définissons une fonction miroir qui prend en entrée et retourne une expression régulière telle que le langage dénoté par cette expression régulière soit le langage miroir de l'expression régulière passée en entrée. Soit Σ un alphabet. Nous définissons la fonction miroir : $ER(\Sigma) \rightarrow ER(\Sigma)$ par induction sur l'expression régulière d'entrée comme suit :

$\text{— miroir}(\emptyset) = \emptyset$ $\text{— miroir}(\epsilon) = \epsilon$ $\text{— miroir}(s) = s$, pour $s \in \Sigma$	$\text{— miroir}(e \cdot f) = \text{miroir}(e) \cdot \text{miroir}(f)$ $\text{— miroir}(e + f) = \text{miroir}(e) + \text{miroir}(f)$ $\text{— miroir}(e^*) = (\text{miroir}(e))^*$
--------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2. Nous démontrons que langage dénoté par l'expression régulière retournée par la fonction est le langage miroir de l'expression régulière passée en entrée ; ceci par induction sur les expressions régulières (domaine de la fonction miroir). Plus précisément, nous démontrons $\forall e \in ER. \mathcal{L}_{ER}(\text{miroir}(e)) = (\mathcal{L}_{ER}(e))^R$, par induction sur e .

- Cas \emptyset . Ce cas est évident.
- Cas ϵ . Nous avons $\text{miroir}(\epsilon) = \epsilon$ et $\mathcal{L}_{ER}(\epsilon) = \{\epsilon\} = \{\epsilon\}^R$.
- Cas $s \in \Sigma$. Ce cas est similaire au cas précédent.
- Cas $e \cdot f$. L'hypothèse d'induction est que le résultat est valide pour deux expressions régulières e et f . Nous avons :

$$\begin{aligned}
 \mathcal{L}_{ER}(\text{miroir}((e \cdot f))) &= \mathcal{L}_{ER}(\text{miroir}(e) \cdot \text{miroir}(f)) \\
 &\quad (\text{définition de la fonction miroir}) \\
 &= \mathcal{L}_{ER}(\text{miroir}(f)) \cdot \mathcal{L}_{ER}(\text{miroir}(e)) \\
 &\quad (\text{sémantique des expressions régulières}) \\
 &= \mathcal{L}_{ER}(f)^R \cdot \mathcal{L}_{ER}(e)^R \\
 &\quad (\text{hypothèse d'induction}) \\
 &= \left\{ w_f \cdot w_e \mid w_f \in \mathcal{L}_{ER}(f)^R \wedge w_e \in \mathcal{L}_{ER}(e)^R \right\} \\
 &\quad (\text{déf. de la concaténation de langages}) \\
 &= \left\{ w_f^R \cdot w_e^R \mid w_f \in \mathcal{L}_{ER}(f) \wedge w_e \in \mathcal{L}_{ER}(e) \right\} \\
 &\quad (\text{déf. du langage miroir}) \\
 &= \left\{ (w_e \cdot w_f)^R \mid w_f \in \mathcal{L}_{ER}(f) \wedge w_e \in \mathcal{L}_{ER}(e) \right\} \\
 &\quad (\text{résultat de l'exercice 101}) \\
 &= \{w_e \cdot w_f \mid w_f \in \mathcal{L}_{ER}(f) \wedge w_e \in \mathcal{L}_{ER}(e)\}^R \\
 &= (\mathcal{L}_{ER}(e) \cdot \mathcal{L}_{ER}(f))^R \\
 &= (\mathcal{L}_{ER}(e \cdot f))^R \\
 &\quad (\text{sémantique des expressions régulières})
 \end{aligned}$$

- Cas $e + f$. L'hypothèse d'induction est que le résultat est valide pour deux expressions régulières e et f . Nous avons :

$$\begin{aligned}
 \mathcal{L}_{ER}(\text{miroir}((e + f))) &= \mathcal{L}_{ER}(\text{miroir}(e) + \text{miroir}(f)) \\
 &\quad (\text{définition de la fonction miroir}) \\
 &= \mathcal{L}_{ER}(\text{miroir}(f)) \cup \mathcal{L}_{ER}(\text{miroir}(e)) \\
 &\quad (\text{sémantique des expressions régulières}) \\
 &= \mathcal{L}_{ER}(f)^R \cup \mathcal{L}_{ER}(e)^R \\
 &\quad (\text{hypothèse d'induction}) \\
 &= (\mathcal{L}_{ER}(f) \cup \mathcal{L}_{ER}(e))^R \\
 &\quad ((X \cup Y)^R = X^R \cup Y^R) \\
 &= (\mathcal{L}_{ER}(f + e))^R \\
 &\quad (\text{sémantique des expressions régulières}) \\
 &= (\mathcal{L}_{ER}(e + f))^R \\
 &\quad (\text{commutativité})
 \end{aligned}$$

— Cas e^* . L'hypothèse d'induction est que le résultat est valide pour une expression régulière e . Nous avons :

$$\begin{aligned}
 \mathcal{L}_{\text{ER}}(\text{miroir}((e^*))) &= \mathcal{L}_{\text{ER}}((\text{miroir}(e))^*) \\
 &\quad (\text{définition de la fonction miroir}) \\
 &= (\mathcal{L}_{\text{ER}}(\text{miroir}(e)))^* \\
 &\quad (\text{sémantique des expressions régulières}) \\
 &= (\mathcal{L}_{\text{ER}}(e)^R)^* \\
 &\quad (\text{hypothèse d'induction}) \\
 &= \left\{ w_1 \cdots w_n \mid n \in \mathbb{N} \wedge w_i \in \mathcal{L}_{\text{ER}}(e)^R \right\} \\
 &\quad (\text{def. de la fermeture de Kleene}) \\
 &= \left\{ w_1^R \cdots w_n^R \mid n \in \mathbb{N} \wedge w_i \in \mathcal{L}_{\text{ER}}(e) \right\} \\
 &= \left\{ (w_n \cdots w_1)^R \mid n \in \mathbb{N} \wedge w_i \in \mathcal{L}_{\text{ER}}(e) \right\} \\
 &\quad (\text{résultat de l'exercice 101}) \\
 &= \left\{ w_n \cdots w_1 \mid n \in \mathbb{N} \wedge w_i \in \mathcal{L}_{\text{ER}}(e) \right\}^R \\
 &= \left\{ w_1 \cdots w_n \mid n \in \mathbb{N} \wedge w_i \in \mathcal{L}_{\text{ER}}(e) \right\}^R \\
 &= (\mathcal{L}_{\text{ER}}(e)^*)^R \\
 &\quad (\text{def. de la fermeture de Kleene}) \\
 &= (\mathcal{L}_{\text{ER}}(e^*))^R \\
 &\quad (\text{sémantique des expressions régulières})
 \end{aligned}$$

Théorème de Kleene

10.1 Résumé intuitif du chapitre

Dans le chapitre précédent, nous avons introduit la notion d'expression régulière, d'équivalence entre expressions régulières. Dans ce chapitre, nous poursuivons cette voie en étudiant les correspondances, d'une part expression régulière et automate et d'autre part entre langage régulier et langage à états. Ces correspondances sont données par le **théorème de Kleene**. Elles expriment le fait que la classe des langages dénotés par une expression régulière est identique à la classe des langages reconnus par un automate.

De plus, pour tout langage associé à une expression régulière, il existe un automate reconnaissant ce langage, et inversement, qu'étant donné un automate il est possible de construire une expression régulière dénotant le langage accepté par un automate. Ceci permet de **décider de l'équivalence et de l'inclusion de langages** entre (langages dénotés par) deux expressions régulières, là où les identités énoncées au chapitre précédent (section 9.2.5) ne suffisent pas. En effet, pour décider de l'équivalence de deux expressions régulières, il suffit d'associer à chacune l'automate qui les caractérise, puis d'utiliser une des méthodes (section 6.2.4) pour déterminer l'équivalence entre deux automates (par exemple par déterminisation, minimisation et test d'existence d'une fonction de renommage). Pour associer une expression régulière à un automate, on distingue principalement deux méthodes :

- dans l'une, on associe des **équations à chaque état** de l'automate, puis on les résout en utilisant notamment le **lemme d'Arden** permettant la résolution d'équations dont les variables sont des langages ;
- dans l'autre, on associe des **équations à des chemins** de longueur croissante entre deux états que l'on peut résoudre itérativement.

Dans le premier cas, chaque variable d'équation caractérise un langage et on s'intéresse au langage associé à la variable de l'état initial. Dans le second cas, chaque variable caractérise un chemin entre deux états et on s'intéresse aux chemins allant de l'état initial à un état accepteur. Nous étudions également une méthode de calcul d'expression régulière associée à un automate qui procède par **élimination des états**. La méthode commence par normaliser l'automate

en le transformant pour assurer (entre autre) qu'il possède un unique état accepteur. Dans cette méthode, des expressions régulières étiquettent les transitions de l'automate. Ensuite, la méthode supprime chacun des états de l'automate jusqu'à que l'automate ne possède plus qu'un état initial et son état accepteur.

Pour associer un automate à une expression régulière, une méthode procède par **induction sur la syntaxe** des expressions régulières. En utilisant la structure de l'expression régulière, la méthode construit de manière **compositionnelle** un automate : à partir d'automates pour les opérandes d'un opérateur, on construit un automate pour l'expression où l'opérateur est appliqué aux opérandes.

10.2 Les notions essentielles

10.2.1 Théorème de Kleene

Théorème 12 (Kleene) Soit $L \subseteq \Sigma^*$ un langage sur un alphabet Σ . L est à états si et seulement si L est régulier.

De plus :

1. On peut algorithmiquement transformer un automate fini en une expression régulière tel que le langage dénoté par cette expression régulière soit le langage reconnu par l'automate.
2. Inversement, on peut algorithmiquement transformer une expression régulière en un automate fini qui reconnaît le langage dénoté par cette expression régulière.

En conséquence du théorème de Kleene, les problèmes de l'équivalence et de l'inclusion entre langages dénotés par des expressions régulières sont décidables.

Corollaire 6 (Décidabilité de l'équivalence et de l'inclusion des langages dénotés) Les problèmes de décision de l'équivalence et de l'inclusion de langages dénotés par des expressions régulières, définitions [131](#) et [132](#), sont décidables.

10.2.2 Des automates vers les expressions régulières

Les méthodes suivantes prennent en entrée un automate A et produisent une expression régulière e_A telle que $\mathcal{L}_{\text{ER}}(e_A) = \mathcal{L}_{\text{auto}}(A)$. Ces méthodes fonctionnent avec différents sorts d'automates (AFED, AFEND, ϵ -AFEND). Nous indiquons pour chaque méthode avec quelle sorte d'automates elle fonctionne. Nous comparons les avantages et inconvénients de ces techniques dans le tableau [10.1](#) (p. [221](#)).

Méthode par calcul des langages associés aux états (J. A. Brzozowski et E. J. McCluskey, 1964)

Soit $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ un AFED ou AFEND.

Tableau 10.1 – Comparaison des méthodes pour passer d'un automate à une expression régulière équivalente.

Méthode	Association d'équations aux états	Élimination des états	Association d'équations aux chemins
Avantages	<ul style="list-style-type: none"> • élégance • génère des expressions régulières raisonnablement compacte 	<ul style="list-style-type: none"> • intuitive • pratique pour la vérification manuelle 	<ul style="list-style-type: none"> • implémentation simple
Inconvénients	<ul style="list-style-type: none"> • pas aussi simple à implémenter que les autres méthodes 	<ul style="list-style-type: none"> • tendance à créer des expressions régulières très longues 	<ul style="list-style-type: none"> • fastidieuse manuellement

Définition 133 (Système d'équations associé à un automate) *Le système d'équations associé à A est l'ensemble d'équations $SE(A) = \{E_q \mid q \in Q\}$ où E_q est l'équation associée à l'état q définie par :*

$$X_q = \sum_{\Delta(q,a)=q} a \cdot X_{q'} + (\text{si } q \in F \text{ alors } \epsilon \text{ sinon } \emptyset)$$

qui fait référence à $X_{q'}$, avec $q' \in Q$, le langage reconnu à partir de l'état q' .

Lemme 5 (Résolution d'équations linéaires, lemme d'Arden) *Soient $E, F \subseteq \Sigma^*$ des langages. Considérons l'équation entre langages $X = E \cdot X + F$. Alors :*

1. *Le langage $E^* \cdot F$ est une solution de l'équation.*
2. *(Lemme d'Arden) : Si $\epsilon \notin E$, alors $E^* \cdot F$ est la solution unique.*

Remarque 40 *Pour résoudre un système d'équations correspondant à un automate, on applique le lemme 5 uniquement dans le deuxième cas.*

Remarque 41 *Comme l'automate d'entrée est soit un AFED ou un AFEND (et non pas un ϵ -AFEND), le deuxième cas du lemme 5 s'applique toujours.*

Théorème 13 (Langage de l'automate) *Si $\{L_{X_q} \mid q \in Q\}$ est la plus petite solution de $SE(A)$ (où L_{X_q} est le langage solution de l'équation associée à l'état $q \in Q$). Alors, $\mathcal{L}_{\text{auto}}(A) = L_{X_{q_0}}$.*

Méthode par élimination des états

Soit $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ un AFED, AFEND ou ϵ -AFEND.

Les deux idées sous-jacentes de la méthode sont d'étiqueter les transitions par des expressions régulières et de supprimer les états (non initiaux et finaux) en mettant à jour les

transitions sans modifier le langage. La technique d'élimination des états nécessite un automate normalisé.

Définition 134 (Automate normalisé) *L'automate A est normalisé si :*

- $\Delta \cap (Q \times \Sigma \times \{q_{\text{init}}\}) = \emptyset$;
- $|F| = 1 \wedge (\Delta \cap (F \times \Sigma \times Q)) = \emptyset$.

La première condition indique que son état initial est sans transition entrante. La seconde condition indique que l'automate possède un seul état final sans transition sortante.

La méthode procède en deux étapes : normalisation de l'automate puis élimination des états.

Définition 135 (Normalisation - pour l'initialisation) *S'ils existent $q \in Q$ et $s \in \Sigma$ tels que $(q, s, q_{\text{init}}) \in \Delta$, alors la version normalisée de A pour l'initialisation est l'automate $(Q \cup \{i\}, \Sigma, i, \Delta \cup \{(i, \epsilon, q_{\text{init}})\}, F)$. Sinon, A est déjà normalisé pour l'initialisation.*

Définition 136 (Normalisation - pour la terminaison) *Si $|F| > 1$ ou s'ils existent $q \in F$, $q' \in Q$ et $s \in \Sigma$ tels que $(q, s, q') \in \Delta$, alors la version normalisée de A pour la terminaison est $(Q \cup \{f\}, \Sigma, q_{\text{init}}, \Delta \cup \{(q, \epsilon, f) \mid q \in F\}, \{f\})$. Sinon, A est déjà normalisé pour la terminaison.*

Définition 137 (Normalisation) *La normalisation d'un automate consiste à lui appliquer successivement les normalisations pour l'initialisation et pour la terminaison. L'automate résultant est normalisé.*

Soit $(Q, \Sigma, i, \Delta, \{f\})$ l'automate résultant de la normalisation de A.

Définition 138 (Algorithme d'élimination des états) *L'algorithme 40 calcule une expression régulière associée à un automate normalisé $(Q, \Sigma, i, \Delta, \{f\})$ passé en paramètre. L'algorithme fonctionne par élimination successive des états, jusqu'à que l'automate ne contienne plus qu'un état initial et un état final. L'algorithme utilise la fonction $R : Q \times Q \rightarrow ER(\Sigma)$ telle que, pour deux états $q, q' \in Q$, $R(q, q')$ est une expression régulière dont le langage dénoté est l'ensemble des mots permettant d'aller de l'état q à l'état q' en suivant la relation de transition de l'automate Δ .*

Soit $R_{q, q'}$ l'expression régulière associée à la transition entre les états q et q' . L'initialisation de l'algorithme consiste à associer à $R(q, q')$ l'expression régulière qui est la somme des symboles étiquetant une transition directe dans Δ entre les états q et q' . Lors de chaque itération de l'algorithme, on choisit un état q qui sera supprimé à la fin de l'itération ($Q := Q \setminus \{q\}$). De plus, l'algorithme met à jour l'expression régulière $R(q_1, q_2)$ pour tous les états dans $q_1, q_2 \in Q \setminus \{q\}$ en y stockant l'expression régulière dénotant l'ensemble des chemins permettant d'aller de q_1 à q_2 en passant par l'état q . Cette nouvelle expression associée à $R(q_1, q_2)$ est $R(q_1, q_2) + R(q_1, q) \cdot R(q, q')^* \cdot R(q, q_2)$, c'est-à-dire que c'est la somme entre l'expression $R(q_1, q_2)$ encodant les mots permettant d'aller de q_1 à q_2 (déjà connus et ne passant pas par l'état q) et l'expression $R(q_1, q) \cdot R(q, q')^* \cdot R(q, q_2)$ qui dénote les mots permettant d'aller de q_1 vers q_2 en passant par l'état q . Ces derniers mots sont la concaténation d'un mot permettant d'aller de q_1 à q , d'une concaténation de mots permettant d'aller de q à q , et d'un mot permettant d'aller de q à q_2 .

Quand l'itération termine, l'algorithme retourne l'expression régulière $R(i, f)$ qui dénote l'ensemble des chemins permettant d'aller de l'état initial i à l'(unique) état final f .

Algorithme 40 Algorithme d'élimination des états

Entrée : $A = (Q, \Sigma, i, \Delta, \{f\})$ (* un ϵ -AFEND normalisé *)

Sortie : e_A (* une expression régulière telle que $\mathcal{L}_{ER}(e_A) = \mathcal{L}_{auto}(A)$ *)

- 1: **fonction** $R : Q \times Q \rightarrow ER(\Sigma)$ (* fonction stockant les étiquettes sur les transitions *)
- 2: **pour** $q \in Q$ faire
- 3: **pour** $q' \in Q$ faire
- 4: $R(q, q') := \sum_{(q, s, q') \in \Delta} s$
- 5: (* l'étiquette est la somme des symboles étiquetant une transition entre q et q' dans Δ *)
- 6: **fin pour**
- 7: **fin pour**
- 8: **tant que** $Q \neq \{i, f\}$ faire
- 9: **soit** $q \in Q \setminus \{i, f\}$; (* on choisit un état q qui va être supprimé *)
- 10: **pour** $q_1 \in Q \setminus \{q\}$ faire
- 11: **pour** $q_2 \in Q \setminus \{q\}$ faire
- 12: $R(q_1, q_2) := R(q_1, q_2) + R(q_1, q) \cdot R(q, q) \cdot R(q, q_2);$
- 13: **fin pour**
- 14: **fin pour**
- 15: **fin tant que**
- 16: **retourner** $R(i, f);$

Remarque 42 L'ordre d'élimination des états influe sur la taille de l'expression finale générée. Des heuristiques existent pour l'ordre dans lequel se fait le choix de l'état à supprimer.

Méthode par calcul des langages associés aux chemins (McNaughton et Yamada (1960))

La méthode construit une collection d'expressions régulières qui décrivent progressivement des chemins de moins en moins contraints dans l'automate, par induction.

Supposons, quitte à utiliser une fonction de renommage, que les états sont numérotés de 1 à n .

Définition 139 (Expression régulière associée à un chemin) Nous définissons $R_{i,j}^k$, l'expression régulière des mots étiquettes d'un chemin entre l'état i et l'état j qui passe uniquement par des états intermédiaires avec un numéro inférieur (strictement) à k . Il n'y a aucune contrainte sur i ni sur j .

- $R_{i,j}^0$ est l'expression régulière des chemins entre l'état i et l'état j dont tous les états intermédiaires ont un numéro plus petit que 0 ; c'est-à-dire qui n'ont aucun état intermédiaire. Pour les chemins directs entre un état i et un état j , il y a deux cas possibles :
 - Si $i \neq j$, alors on regarde les transitions directes entre i et j . S'il n'y a pas de transition, alors $R_{i,j}^0 = \emptyset$. S'il y a une transition étiquetée par un symbole s , alors $R_{i,j}^0 = s$. Sinon, il y a plusieurs transitions étiquetées par des symboles s_1, \dots, s_n , alors $R_{i,j}^0 = s_1 + \dots + s_n$. Ainsi, lorsque $i \neq j$, on a :

$$R_{i,j}^0 = \sum_{(i,s,j) \in \Delta} s.$$

- Sinon ($i = j$), les cas précédents s'appliquent de la même manière. Il faut de plus ajouter ϵ à chaque expression régulière. Ainsi, lorsque $i = j$, on a :

$$R_{i,i}^0 = \epsilon + \sum_{(i,s,i) \in \Delta} s.$$

- Considérons un chemin de $R_{i,j}^k$:

- Soit il ne passe pas par l'état k , alors c'est un chemin de $R_{i,j}^{k-1}$.
- Soit il passe par l'état k (au moins une fois). On peut décomposer ce chemin en chemins qui ne passent pas par un état intermédiaire plus grand que $k - 1$.

Ainsi, l'expression régulière associée à $R_{i,j}^k$ peut être définie en fonction de $R_{i,j}^{k-1}$, $R_{i,k}^{k-1}$, $R_{k,k}^{k-1}$ et $R_{k,j}^{k-1}$ comme suit :

$$R_{i,j}^k = R_{i,j}^{k-1} + R_{i,k}^{k-1} \cdot \left(R_{k,k}^{k-1} \right)^* \cdot R_{k,j}^{k-1}.$$

Définition 140 (Méthode de McNaughton et Yamada) La méthode utilise les étapes suivantes :

1. Renommer les états de l'automate pour qu'ils soient numérotés de 1 à $|Q|$.
2. Calculer les $R_{i,j}^0$, pour $i, j \in [1, |Q|]$.
3. Calculer les $R_{i,j}^k$, $i, j, k \in [1, |Q|]$, en utilisant l'expression récursive de $R_{i,j}^k$.

L'expression régulière de l'automate est alors :

$$\sum_{f \in F} R_{1,f}^n.$$

Remarque 43 En pratique, on arrêtera le calcul à $R_{i,j}^{|Q|-1}$ et calculera uniquement les expressions régulières $R_{1,f}^{|Q|}$, pour $f \in F$.

10.2.3 Des expressions régulières vers les automates

Les méthodes suivantes prennent en entrée une expression régulière e et construisent un automate A_e tel que $\mathcal{L}_{\text{auto}}(A_e) = \mathcal{L}_{\text{ER}}(e)$.

Remarque 44 Pour obtenir un automate à partir d'une expression régulière, nous pourrons également nous appuyer sur la sémantique des expressions régulières et celle des ϵ -AFEND pour écrire directement un ϵ -AFEND. C'est le cas dans l'exercice 126 (p. 232).

Méthode compositionnelle

Cette méthode construit des automates par induction sur la syntaxe de l'expression régulière considérée en distinguant :

- les expression régulière de base : ϵ , \emptyset , symboles de l'alphabet ;

- les *expressions régulières composées* : union, concaténation, fermeture de Kleene ; les constructions associées se font en fonction d'automates définis pour les expression régulières opérandes.

Cette méthode utilise des automates normalisés. Nous représentons les automates de manière graphique et schématique comme illustré dans la figure 10.1.

Définition 141 (Méthode de construction compositionnelle d'un ϵ -AFEND) La méthode de construction compositionnelle d' ϵ -AFEND à partir d'expressions régulières est définie de manière inductive en suivant la structure syntaxique de l'expression régulière à laquelle elle s'applique :

- Pour les expressions régulières de base, les constructions sont décrites de manière schématique dans la figure 10.2 (p. 226).
 - Pour l'expression régulière \emptyset , l'automate est décrit dans la figure 10.2a.
 - Pour l'expression régulière ϵ , l'automate est décrit dans la figure 10.2b.
 - Pour l'expression régulière qui est un symbole $s \in \Sigma$, l'automate est décrit dans la figure 10.2c.
- Pour les expressions régulières composées, les constructions sont décrites de manière schématique dans la figure 10.3 (p. 226). Les constructions sont faites à partir de deux automates normalisés A et B pour deux expressions régulières e_A et e_B représentés dans la figure 10.3a.
 - Pour la somme $e_A + e_B$, l'automate obtenu est décrit dans la figure 10.3b.
 - Pour la concaténation $e_A \cdot e_B$, l'automate obtenu est décrit dans la figure 10.3c.
 - Pour la fermeture de Kleene e_A^* , l'automate obtenu est décrit dans la figure 10.3d.

Remarque 45 Définir formellement les générations/transformations d'automates décrites dans la définition 141 fait l'objet de l'exercice 123.

Propriété 13 (Normalisation des automates) Tout automate obtenu/construit à partir de la méthode décrite dans la définition 141 est normalisé.

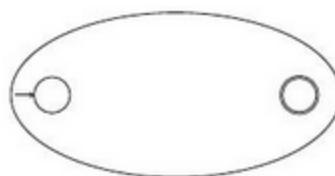


Figure 10.1 – Représentation schématique des automates normalisés utilisés dans la méthode compositionnelle de calcul d'automates à partir d'expression régulières. L'ellipse englobante représente l'ensemble/espace d'états de l'automate. L'état initial et l'unique état accepteur sont représentés comme précédemment dans l'espace d'états.

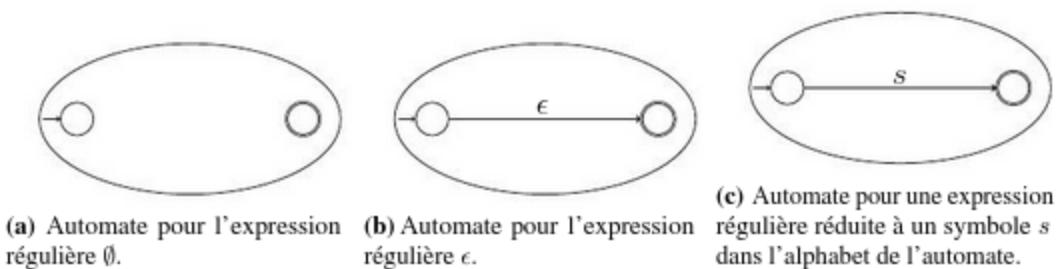


Figure 10.2 – Automates normalisés pour la méthode de construction compositionnelle à partir d'expressions régulières - expressions régulières de base.

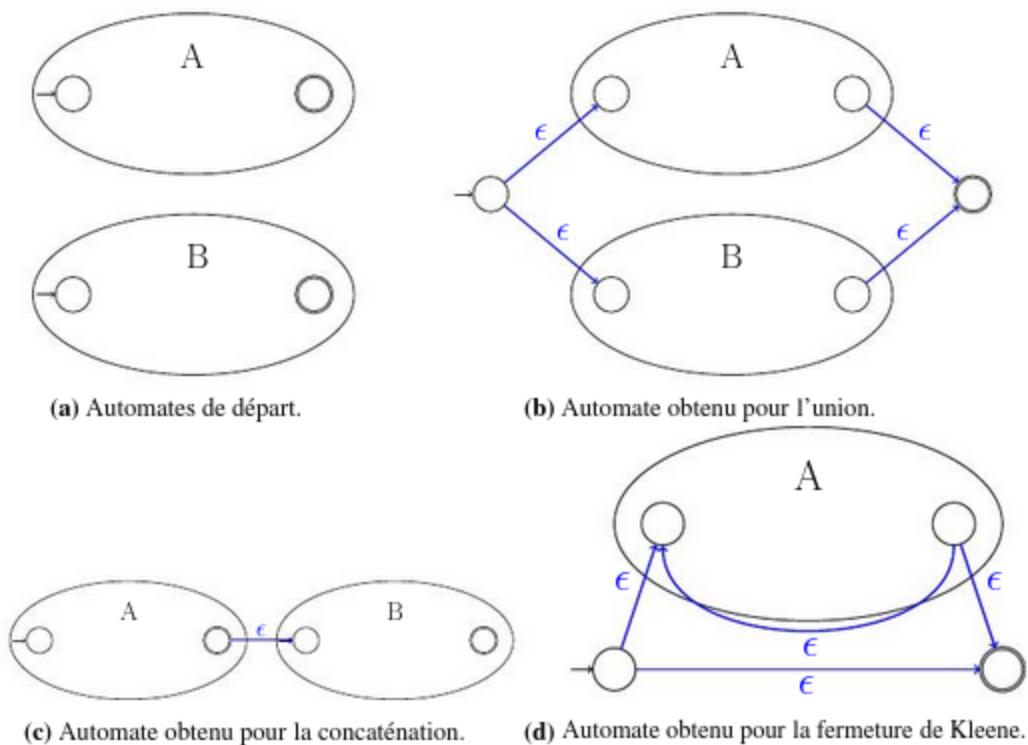


Figure 10.3 – Automates normalisés pour la méthode de construction compositionnelle à partir d'expressions régulières - expressions régulières composées.

Méthode par calcul des dérivées

La méthode a été introduite par J. Brzozowski (concernant la notion de dérivée d'une expression régulière) (1964) et Antimirov (concernant la notion de dérivée partielle d'une expression régulière) (1996). C'est une méthode purement algébrique ne nécessitant pas de construction explicite de l'automate.

Cette méthode fonctionne par calcul de la dérivée d'une expression régulière pour laquelle on veut construire un automate. Intuitivement, la dérivée d'une expression régulière e sur un symbole a est une expression régulière décrivant « ce qu'il manque » après avoir lu a pour former un mot de e .

Les avantages de cette méthode sont d'une part qu'elle utilise uniquement la *syntaxe* des expressions régulières et d'autre part qu'elle peut se faire “*à la volée*” (sans construire entièrement l'automate).

Définition 142 (Terme constant d'une expression régulière) *Le terme constant d'une expression régulière est une expression régulière obtenue par la fonction $c : ER \rightarrow \{\epsilon, \emptyset\}$ définie par :*

$$c(e) = \begin{cases} \epsilon & \text{si } \epsilon \in \mathcal{L}_{ER}(e), \\ \emptyset & \text{sinon.} \end{cases}$$

Propriété 14 (Calcul du terme constant d'une expression régulière) *Afin de pouvoir être calculé directement à partir de l'expression régulière, le terme constant peut être également défini/calculé inductivement comme suit :*

- $c(\emptyset) = \emptyset,$
- $c(\epsilon) = \epsilon,$
- $c(s) = \emptyset, \text{ pour } s \in \Sigma,$
- $c(e + e') = c(e) + c(e'),$
- $c(e \cdot e') = c(e) \cdot c(e'),$
- $c(e^*) = \epsilon.$

Remarque 46 *Calculer le terme constant suppose de simplifier l'expression régulière résultat (dans le cas où il est calculé pour des expressions régulières composées).*

Définition 143 (Dérivée d'une expression régulière par rapport à un symbole) *La dérivée de $e \in ER$ par rapport à $s \in \Sigma$ est l'expression régulière notée $\frac{\partial}{\partial s} e$ et dénotant le langage $\{w \in \Sigma^* \mid s \cdot w \in \mathcal{L}_{ER}(e)\}$.*

Intuitivement, la dérivée d'une expression régulière e sur un symbole s est une expression régulière décrivant ce qu'il manque après avoir lu s pour former un mot dans l'ensemble des mots dénoté par e .

Nous donnons la précédence à l'opérateur de dérivation par rapport aux autres opérateurs.

Propriété 15 (Calcul de la dérivée d'une expression régulière par rapport à un symbole) *La dérivée par rapport à un symbole $s \in \Sigma$ est définie inductivement sur la syntaxe des expressions régulières comme suit :*

- $\frac{\partial}{\partial s} \emptyset = \emptyset,$
- $\frac{\partial}{\partial s} \epsilon = \emptyset,$
- $\frac{\partial}{\partial s} s' = \epsilon, \text{ lorsque } s' = s,$
- $\frac{\partial}{\partial s} s' = \emptyset, \text{ lorsque } s' \neq s,$
- $\frac{\partial}{\partial s} (e + e') = \frac{\partial}{\partial s} e + \frac{\partial}{\partial s} e',$
- $\frac{\partial}{\partial s} (e \cdot e') = \frac{\partial}{\partial s} e \cdot e' + c(e) \cdot \frac{\partial}{\partial s} e',$
- $\frac{\partial}{\partial s} e^* = \frac{\partial}{\partial s} e \cdot e^*.$

La dérivée d'une expression régulière étant une expression régulière, nous pouvons voir $\frac{\partial}{\partial s} e$, la dérivée d'une expression régulière e par rapport à un symbole s , comme l'application d'un opérateur de dérivation $\frac{\partial}{\partial s}$ à l'expression régulière e . Nous donnons à cet opérateur la précédence sur les opérateurs de concaténation et d'union, mais pas sur l'itération. Ainsi, $\frac{\partial}{\partial s} e_1 + e_2$, $\frac{\partial}{\partial s} e_1 \cdot e_2$ et $\frac{\partial}{\partial s} e_1^*$ s'interprètent comme $(\frac{\partial}{\partial s} e_1) + e_2$, $(\frac{\partial}{\partial s} e_1) \cdot e_2$ et $\frac{\partial}{\partial s}(e_1^*)$, respectivement.

Définition 144 (Dérivée d'une expression régulière par rapport à un mot) La dérivée par rapport à un mot dans Σ^* est défini inductivement sur les mots comme suit :

$$— \frac{\partial}{\partial \epsilon} e = e, \quad — \frac{\partial}{\partial s \cdot u} e = \frac{\partial}{\partial u} \left(\frac{\partial}{\partial s} e \right).$$

Intuitivement, dériver par rapport à un mot consiste à dériver *récursivement* et par rapport à chaque lettre du mot dans l'ordre de lecture de ce mot.

Définition 145 (Ensemble des dérivées) Soit $e \in ER$ une expression régulière définie sur un alphabet Σ , l'ensemble des (expressions régulières) dérivées de e , noté $\partial(e)$, est l'ensemble $\left\{ \frac{\partial}{\partial u} e \mid u \in \Sigma^* \right\} \in \mathcal{P}(ER)$.

Proposition 11 (Finitude de l'ensemble des dérivées) L'ensemble des dérivées d'une expression régulière est fini modulo associativité, commutativité et idempotence des opérateurs.

Définition 146 (Automate des dérivées) L'automate des dérivées associée à e est l'AFED, noté $\text{auto_derivees}(e)$, défini par $(\partial e, \Sigma, e, \delta_e, F_e)$ avec :

- $\delta_e(d, a) = \frac{\partial}{\partial a} d$, pour $d \in \partial e$ et $a \in \Sigma$,
- $F_e = \{d \in \partial e \mid c(d) = \epsilon\}$.

Proposition 12 (Correction de l'automate des dérivées) L'automate des dérivées reconnaît le langage dénoté par l'expression régulière à partir de laquelle il a été généré :

$$\mathcal{L}_{\text{auto}}(\text{auto_derivees}(e)) = \mathcal{L}_{\text{ER}}(e).$$

10.3 Exercices

Exercice 117 (♠♦) — Expressions régulières étendues

L'ensemble des expression régulières étendues est obtenu en ajoutant les constructions suivantes aux expressions régulières telles que définies dans la définition 127 (p. 206) et la définition 128 (p. 206) :

- si e est une expression régulière sur Σ alors $\neg e$ est une expression régulière sur Σ .
- si e et e' sont des expressions régulières sur Σ , alors $e \cap e'$ est une expression régulière sur Σ .
- si e est une expression régulière sur Σ alors e^+ est une expression régulière sur Σ .

La sémantique de ces opérateurs est définie comme suit :

$$\begin{aligned} \text{--- } \mathcal{L}_{\text{ER}}(\neg e) &= \Sigma^* \setminus \mathcal{L}_{\text{ER}}(e). \\ \text{--- } \mathcal{L}_{\text{ER}}(e \cap e') &= \mathcal{L}_{\text{ER}}(e) \cap \mathcal{L}_{\text{ER}}(e'). \end{aligned}$$

$$\text{--- } \mathcal{L}_{\text{ER}}(e^+) = \mathcal{L}_{\text{ER}}(e) \cdot \mathcal{L}_{\text{ER}}(e)^*.$$

1. Donner les grandes lignes d'un algorithme qui transforme toute expression régulière étendue vers une expression régulière.
2. Que peut-on déduire de l'existence de l'algorithme trouvé à la question précédente ?

10.3.1 Calcul d'expressions régulières à partir d'automates

Exercice 118 (♠♠) — Méthode associant des équations aux états

Nous considérons les AFED dans la figure 10.4b (p. 229) et cherchons des expressions régulières qui dénotent les langages reconnus par ces automates, en utilisant la méthode associant des équations aux états.

1. Donner une expression régulière pour l'AFED dans la figure 10.4a.
2. Donner une expression régulière pour l'AFED dans la figure 10.4b.
3. Donner une expression régulière pour l'AFED dans la figure 10.4c.
4. Donner une expression régulière pour l'AFED dans la figure 10.4d.

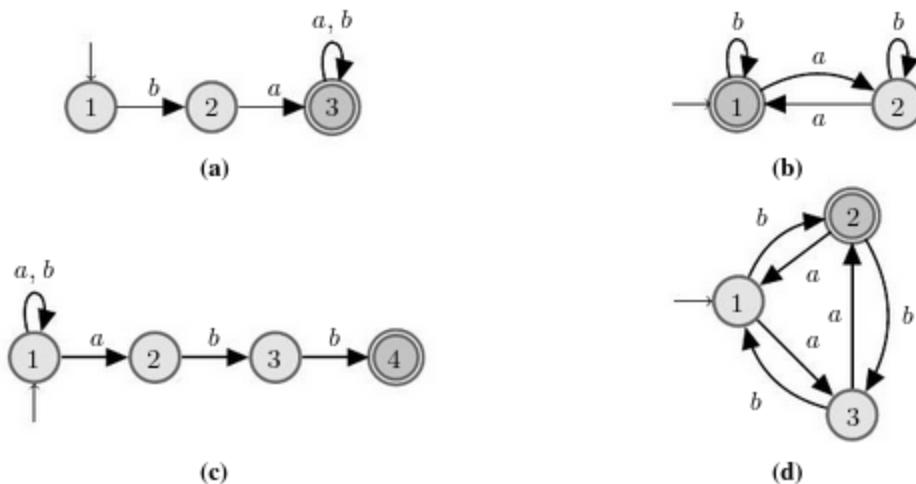
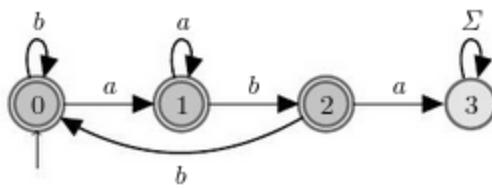
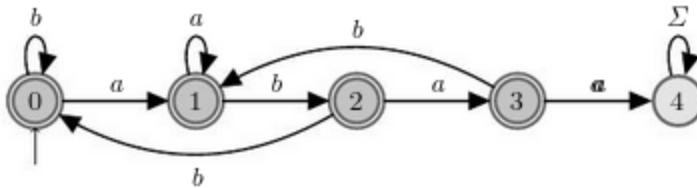


Figure 10.4 – Des automates à utiliser pour le calcul d'expressions régulières et de grammaire (au chapitre 12).

Exercice 119 (♠♠♠) — Méthode associant des équations aux états

Nous considérons quelques un des AFED minimaux du chapitre 6 et rappelés dans la figure 10.5.

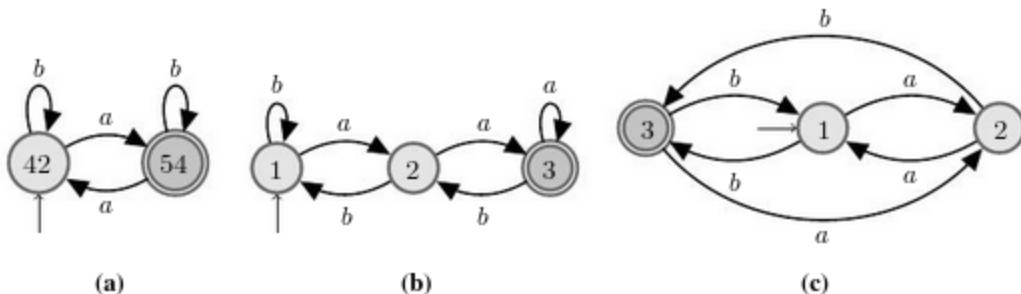
(a) Automate reconnaissant les mots ne contenant pas aba .(b) Automate reconnaissant les mots ne contenant pas $abaa$.**Figure 10.5 – Quelques automates minimaux trouvés au chapitre 6.**

- Pour chaque automate, donner une expression qui dénote le langage qu'il reconnaît en utilisant la méthode associant des équations aux états.

Exercice 120 (♦♦) — Méthode associant des équations aux chemins

Nous souhaitons calculer les expressions régulières associées aux AFED dans la figure 10.6 en suivant la méthode associant des expressions régulières aux chemins.

- Calculer les expressions régulières pour l'automate de la figure 10.6a.
- Calculer les expressions régulières pour l'automate de la figure 10.6b.
- Calculer les expressions régulières pour l'automate de la figure 10.6c.

**Figure 10.6 – Des automates pour le calcul d'expressions régulières.**

Exercice 121 (♦♦) — Méthode par élimination des états

Nous souhaitons calculer les expressions régulières associées aux AFED dans la figure 10.7 en suivant la méthode par élimination des états.

1. Pour chacun des automates, indiquer s'il est normalisé. Si cela n'est pas le cas, indiquer les transitions causant la non-normalisation puis normaliser ces automates.
 2. Donner une expression régulière dont langage dénoté est le langage reconnu par l'automate en utilisant la méthode par élimination des états.

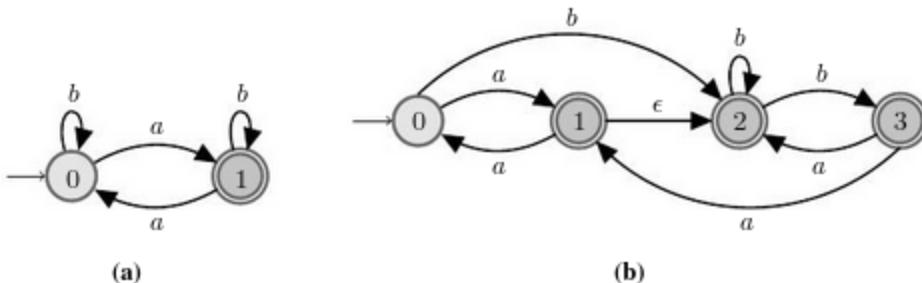


Figure 10.7 – Automates à utiliser pour le calcul d'expression régulière en utilisant la méthode par élimination des états dans l'exercice 121.

Exercice 122 (♠♠) — Démonstration du lemme d'Arden

Soient $E, F, X \subseteq \Sigma^*$ des langages sur un alphabet Σ .

- Démontrer que le langage E^*F est une solution de l'équation $X = EX + F$.
 - Démontrer que si $\epsilon \notin E$, alors E^*F est la solution unique de $X = EX + F$.

10.3.2 Calcul d'automates à partir d'expressions régulières

Exercice 123 (♠♠) — Formalisation de la méthode compositionnelle

1. Formaliser les transformations d'automates décrites dans la définition 141 (p. 225).

Exercice 124 (♠♠) — En utilisant la méthode compositionnelle

Soit $\Sigma = \{a, b\}$. Donner les e -AFEND associés aux expressions régulières suivantes en utilisant la méthode compositionnelle de Thompson.

- $$1. \ a \cdot b, \quad 2. \ a^* \cdot b, \quad 3. \ (a+b)^* \cdot a^* \cdot b^*.$$

Exercice 125 (♠♠) — En utilisant la méthode par calcul des dérivées

Soit $\Sigma = \{a, b\}$. Donner les AFED associés aux expressions régulières suivantes en utilisant la méthode par calcul des dérivées.

- | | |
|--------------------|----------------------------------------|
| 1. $a \cdot b$. | 3. $(a + b)^* \cdot a$. |
| 2. $a^* \cdot b$. | 4. $a^* \cdot b + a \cdot (a + b)^*$. |

Exercice 126 (♠♦) — En utilisant la sémantique des expressions régulières

Soit $\Sigma = \{a, b\}$. On cherche les ϵ -AFEND associés aux expressions régulières suivantes. Cependant, l'utilisation des méthodes présentées dans la section 10.2.3 est trop fastidieuse pour ces expressions régulières. Utiliser la sémantique des expressions régulières et celles des ϵ -AFEND pour obtenir directement les ϵ -AFEND correspondants.

1. $(a^* \cdot b + d \cdot c)^* \cdot (b^* \cdot d + a \cdot d)^*$.
2. $(a \cdot b + a^* \cdot b + c \cdot d) \cdot ((c \cdot a^* + b \cdot d) \cdot (a \cdot b^* + a \cdot b \cdot d))^*$.

10.4 Indications pour résoudre les exercices

Indications pour l'exercice 117 (p. 228)

1. Utiliser les traductions entre expressions régulières et automates.

Indications pour l'exercice 118 (p. 229)

1. Pour chaque question, traduire les automates en système d'équations et utiliser le lemme d'Arden au besoin. Pour chaque application du lemme d'Arden, vérifier que la condition pour avoir une solution unique est vérifiée. L'expression régulière associée à l'automate est la solution trouvée pour l'équation associée à l'état initial.

Pour cet automate, appliquer le lemme d'Arden sur l'équation associée à X_3 puis substituer le résultant dans X_2 . Enfin, substituer le résultat dans X_1 .

2. Appliquer le lemme d'Arden sur l'équation associée à X_2 puis substituer le résultat dans l'équation associée à X_1 .
3. Substituer l'équation associée à X_4 dans l'équation associée à X_3 , puis successivement à X_2 et à X_1 .
4. Substituer l'équation associée à X_3 dans l'équation associée à X_1 . Substituer l'équation associée à X_3 dans l'équation associée à X_2 . Appliquer le lemme d'Arden à l'équation trouvée pour X_2 . Substituer l'équation trouvée pour X_2 dans l'équation associée à X_1 . Appliquer le lemme d'Arden à l'équation trouvée pour X_1 .

Indications pour l'exercice 119 (p. 229)

1. Suivre le même principe que pour l'exercice 118. Traduire les automates en systèmes d'équations.

- Pour l'automate dans la figure 10.5a, appliquer le lemme d'Arden sur l'équation de l'état 3, puis celle de l'état 1, puis celle de l'état 0.
- Pour l'automate dans la figure 10.5b, appliquer le lemme d'Arden sur l'équation de l'état 4, puis celle de l'état 0.

Appliquer des substitutions au besoin entre les applications du lemme d'Arden.

Indications pour l'exercice 120 (p. 230)

1. Suivre la méthode, en renommant les états de l'automate puis en calculant les $R_{i,j}^k$ pour $k = 0$ jusqu'à $k = |Q|$.

Indications pour l'exercice 121 (p. 230)

1. Les automates ne sont pas normalisés.
2. Supprimer les états l'un après l'autre en modifiant les transitions suivant la méthode du cours à chaque étape.

Indications pour l'exercice 122 (p. 231)

1. Montrer que $X = EX + F$ est vraie lorsqu'on remplace X par E^*F .
2. Faire une preuve par l'absurde en supposant l'existence de deux solutions distinctes. Considérer le plus petit mot qui soit dans une solution et pas dans l'autre pour trouver une contradiction.

10.5 Solutions des exercices

Solution de l'exercice 117 (page 228)

1. Un algorithme s'obtient en composant les conversions et opérations suivantes :
 - expression régulière vers automate, en utilisant l'une des méthodes décrites dans la section 10.2.3 (p. 224),
 - les opérations de complémentation et de produit sur les automates ; voir définition 79 (p. 72) et définition 80 (p. 73), respectivement.
 - automate vers expression régulière, en utilisant l'une des méthodes décrites dans la section 10.2.2 (p. 220).
2. En conséquence de l'existence de cet algorithme, nous déduisons que les expressions régulières étendues permettent de décrire les mêmes langages que les expressions régulières. Toute expression régulière étendue peut ainsi être traduite (automatiquement) en un automate équivalent (c'est-à-dire qui reconnaît le langage dénoté par cette expression régulière étendue).

10.5.1 Calcul d'expressions régulières à partir d'automates

Solution de l'exercice 118 (page 229)

1. Nous calculons une expression régulière pour l'automate représenté dans la figure 10.4a (p. 229). Le système d'équations associé à cet automate est :

$$\begin{cases} X_1 &= b \cdot X_2 \\ X_2 &= a \cdot X_3 \\ X_3 &= (a+b) \cdot X_3 + \epsilon \end{cases}$$

En utilisant les conventions de notations (section 9.2.3), le système peut s'écrire également :

$$\begin{cases} X_1 = bX_2 \\ X_2 = aX_3 \\ X_3 = (a+b)X_3 + \epsilon \end{cases}$$

En appliquant le lemme d'Arden sur l'équation associée à X_3 , comme $\epsilon \notin \mathcal{L}_{\text{ER}}(a+b)$, nous obtenons $X_3 = (a+b)^*$. En substituant l'équation trouvée pour X_3 dans l'équation associée à X_2 , nous obtenons $X_2 = a(a+b)^*$. Finalement, en substituant l'équation trouvée pour X_2 dans l'équation associée à X_1 , nous obtenons $X_1 = ba(a+b)^*$.

- Nous calculons une expression régulière pour l'automate représenté dans la figure 10.4b (p. 229). Le système d'équations associé à cet automate est :

$$\begin{cases} X_1 = aX_2 + bX_1 + \epsilon \\ X_2 = aX_1 + bX_2 \end{cases}$$

En appliquant le lemme d'Arden à l'équation associée à X_2 , puis en substituant le résultat dans l'équation associée à X_1 , nous obtenons :

$$\begin{cases} X_1 = bX_1 + ab^*(aX_1) + \epsilon = (b + ab^*a)X_1 + \epsilon \\ X_2 = b^*(aX_1) \end{cases}$$

Finalement, en appliquant le lemme d'Arden sur l'équation associée à X_1 , comme $\epsilon \notin \mathcal{L}_{\text{ER}}(b + ab^*a)$, nous obtenons $X_1 = (b + ab^*a)^*$.

- Nous calculons une expression régulière pour l'automate représenté dans la figure 10.4c (p. 229). Le système d'équations associé à cet automate est :

$$\begin{cases} X_1 = (a+b)X_1 + aX_2 \\ X_2 = bX_3 \\ X_3 = bX_4 \\ X_4 = \epsilon \end{cases}$$

En substituant l'équation associée à X_4 dans l'équation associée à X_3 , nous obtenons $X_3 = b$. En substituant l'équation associée à X_3 dans l'équation associée à X_2 , nous obtenons $X_2 = bb$. En substituant l'équation associée à X_2 dans l'équation associée à X_1 , nous obtenons $X_1 = (a+b)X_1 + abb$. Finalement, en appliquant le lemme d'Arden sur l'équation associée à X_1 , comme $\epsilon \notin \mathcal{L}_{\text{ER}}(a+b)$, nous obtenons

$$X_1 = (a+b)^*abb.$$

Nous remarquons que cet automate est non déterministe mais que l'application de la méthode reste la même.

- Nous calculons une expression régulière pour l'automate représenté dans la figure 10.4d (p. 229). Le système d'équations associé à cet automate est :

$$\begin{cases} X_1 = bX_2 + aX_3 \\ X_2 = aX_1 + bX_3 + \epsilon \\ X_3 = bX_1 + aX_2 \end{cases}$$

En substituant l'équation associée à X_3 dans l'équation associée à X_1 , nous obtenons $X_1 = bX_2 + abX_1 + aaX_2 = (aa + b)X_2 + abX_1$. En substituant l'équation associée à X_3 dans l'équation associée à X_2 , nous obtenons $X_2 = aX_1 + bbX_1 + baX_2 + \epsilon = (a + bb)X_1 + baX_2 + \epsilon$. En appliquant le lemme d'Arden à X_2 , comme $\epsilon \notin \mathcal{L}_{\text{ER}}(ba)$, nous obtenons $X_2 = (ba)^*((a + bb)X_1 + \epsilon)$. Considérons l'équation associée à X_1 , en utilisant l'équation trouvée pour X_2 , nous trouvons :

$$\begin{aligned} X_1 &= (aa + b)(ba)^*(a + bb)X_1 + (aa + b)(ba)^* + abX_1 \\ &= ((aa + b)(ba)^*(a + bb) + ab)X_1 + (aa + b)(ba)^* \end{aligned}$$

En utilisant le lemme d'Arden sur l'équation trouvée pour X_1 , comme $\epsilon \notin \mathcal{L}_{\text{ER}}((aa + b)(ba)^*(a + bb) + ab)$, nous trouvons :

$$X_1 = ((aa + b)(ba)^*(a + bb) + ab)^*(aa + b)(ba)^*.$$

Solution de l'exercice 119 (page 229)

1. Nous calculons une expression régulière pour chacun des deux automates de la figure 10.5 (p. 230).

— Pour l'automate reconnaissant les mots ne contenant pas aba (figure 10.5a), le système d'équations est donné ci-dessous :

$$\left\{ \begin{array}{l} X_0 = aX_1 + bX_0 + \epsilon \\ X_1 = aX_1 + bX_2 + \epsilon \\ X_2 = aX_3 + bX_0 + \epsilon \\ X_3 = (a + b)X_3 \end{array} \right.$$

En appliquant le lemme d'Arden à l'équation de X_3 (ce qui est possible car $\epsilon \notin a + b$) et en simplifiant l'équation de X_2 , nous obtenons :

$$\left\{ \begin{array}{l} X_0 = aX_1 + bX_0 + \epsilon \\ X_1 = aX_1 + b(bX_0 + \epsilon) + \epsilon \\ X_2 = bX_0 + \epsilon \\ X_3 = \emptyset \end{array} \right.$$

En injectant la définition de X_2 obtenue dans la définition de X_1 puis en appliquant le lemme d'Arden à l'équation de X_1 (ce qui est possible car $\epsilon \notin a$) et en injectant la définition trouvée pour X_0 , nous obtenons :

$$\left\{ \begin{array}{l} X_0 = aa^*(bbX_0 + b + \epsilon) + bX_0 + \epsilon \\ X_1 = a^*(bbX_0 + b + \epsilon) \\ X_2 = bX_0 + \epsilon \\ X_3 = \emptyset \end{array} \right.$$

C'est-à-dire $X_0 = aa^*(bbX_0 + b + \epsilon) + bX_0 + \epsilon = (aa^*bb + b)X_0 + (aa^*(b + \epsilon) + \epsilon)$. En appliquant le lemme d'Arden à l'équation de X_0 (ce qui est possible car $\epsilon \notin \mathcal{L}_{\text{ER}}(aa^*bb + b)$), nous obtenons : $X_0 = (aa^*bb + b)^*(aa^*(b + \epsilon) + \epsilon)$.

Notons que nous pouvons également utiliser les expressions régulières étendues (voir exercice 117) pour simplifier cette expression en :

$$X_0 = \Sigma^* \setminus (\Sigma^* \cdot a \cdot b \cdot a \cdot \Sigma^*).$$

— Pour l'automate reconnaissant les mots ne contenant pas $abaa$ (figure 10.5b), le système d'équations est donné ci-dessous :

$$\left\{ \begin{array}{l} X_0 = 1X_0 + 0X_1 + \epsilon \\ X_1 = 0X_1 + 1X_2 + \epsilon \\ X_2 = 0X_3 + 1X_0 + \epsilon \\ X_3 = 1X_1 + 0X_4 + \epsilon \\ X_4 = (0+1)X_4 \end{array} \right.$$

En appliquant le lemme d'Arden sur l'équation de X_4 (ce qui est possible car $\epsilon \notin \mathcal{L}_{\text{ER}}(0+1)$) et en injectant la définition de X_2 dans l'équation associée à X_1 , nous obtenons :

$$\left\{ \begin{array}{l} X_0 = 1X_0 + 0X_1 + \epsilon \\ X_1 = 0X_1 + 101X_1 + 10 + 11X_0 + 1 + \epsilon \\ X_2 = 01X_1 + 0 + 1X_0 + \epsilon \\ X_3 = 1X_1 + \epsilon \\ X_4 = \emptyset \end{array} \right.$$

Nous réécrivons l'équation $X_1 = 0X_1 + 101X_1 + 10 + 11X_0 + 1 + \epsilon$ en $X_1 = (0+101)X_1 + 10 + 11X_0 + 1 + \epsilon$. En appliquant le lemme d'Arden à l'équation de X_1 (ce qui est possible car $\epsilon \notin \mathcal{L}_{\text{ER}}(0+101)$), nous obtenons : $X_1 = (0+101)^*(10+11X_0+1)$, puis :

$$\begin{aligned} X_0 &= 1X_0 + 0(0+101)^*(10+11X_0+1) + \epsilon \\ X_0 &= 1X_0 + 0(0+101)^*11X_0 + 0(0+101)^*(10+1+\epsilon) + \epsilon \\ X_0 &= 1X_0 + 0(0+101)^*11(0(0+101)^*(10+1+\epsilon) + \epsilon) \end{aligned}$$

Notons que nous pouvons également utiliser les expressions régulières étendues (voir exercice 117) pour simplifier l'expression en :

$$\Sigma^* \setminus \Sigma^* 0100 \Sigma^*.$$

Solution de l'exercice 120 (page 230)

- Pour l'automate dans la figure 10.6a. Nous commençons par renommer les états de l'automate : l'état 42 devient l'état 1, l'état 54 devient l'état 2. Nous calculons l'expression régulière en utilisant la méthode associant des expressions aux chemins de l'automate. C'est-à-dire, nous calculons $R_{i,j}^k$ pour $k \in [0, 2]$, avec $R_{i,j}^k$ comme définie dans les rappels de cours.

— Pour $k = 0$, nous avons :

$$\begin{array}{ll} - R_{1,1}^0 = b + \epsilon, & - R_{2,1}^0 = a, \\ - R_{1,2}^0 = a, & - R_{2,2}^0 = b + \epsilon \end{array}$$

Pour $k = 1$, nous avons :

$$\begin{aligned} &— R_{1,1}^1 = b^*, &— R_{2,1}^1 = a \cdot b^*, \\ &— R_{1,2}^1 = b^* \cdot a, &— R_{2,2}^1 = b + a \cdot b^* \cdot a. \end{aligned}$$

Pour $k = 2$, nous avons :

$$\begin{aligned} R_{1,2}^2 &= R_{1,2}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 \\ &= (b^* \cdot a) + (b^* \cdot a) \cdot (b + a \cdot b^* \cdot a)^* \cdot (b + a \cdot b^* \cdot a) \\ &= (b^* \cdot a) + (b^* \cdot a) \cdot (b + a \cdot b^* \cdot a)^+ \\ &= (b^* \cdot a) \cdot (b + a \cdot b^* \cdot a)^* \end{aligned}$$

2. Pour l'automate dans la figure 10.6b. Nous observons que les états de l'automate sont numérotés à partir de 1. Nous calculons l'expression régulière en utilisant la méthode associant des expressions aux chemins de l'automate. C'est-à-dire, nous calculons $R_{i,j}^k$ pour $k \in [0, 3]$, avec $R_{i,j}^k$ comme définie dans les rappels de cours.

— Pour $k = 0$, nous avons :

$$\begin{array}{lll} — R_{1,1}^0 = \epsilon + b & — R_{2,1}^0 = b & — R_{3,1}^0 = \emptyset \\ — R_{1,2}^0 = a & — R_{2,2}^0 = \epsilon & — R_{3,2}^0 = b \\ — R_{1,3}^0 = \emptyset & — R_{2,3}^0 = a & — R_{3,3}^0 = a + \epsilon \end{array}$$

— Pour $k = 1$, nous avons :

$$\begin{aligned} — R_{1,1}^1 &= R_{1,1}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = (\epsilon + b) \cdot (\epsilon + b) \cdot (\epsilon + b)^* \cdot (\epsilon + b) = b^*, \\ — R_{1,2}^1 &= R_{1,2}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = a + (\epsilon + b) \cdot (\epsilon + b)^* \cdot a = b^* a, \\ — R_{1,3}^1 &= R_{1,3}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = \emptyset + (\epsilon + b) \cdot (\epsilon + b)^* \cdot \emptyset = \emptyset, \\ — R_{2,1}^1 &= R_{2,1}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = b^+, \\ — R_{2,2}^1 &= R_{2,2}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = b^+ a, \\ — R_{2,3}^1 &= R_{2,3}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = a, \\ — R_{3,1}^1 &= R_{3,1}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = \emptyset, \\ — R_{3,2}^1 &= R_{3,2}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = b, \\ — R_{3,3}^1 &= R_{3,3}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = a + \epsilon. \end{aligned}$$

— Pour $k = 2$, nous avons :

$$\begin{aligned} — R_{1,1}^2 &= R_{1,1}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = b^* + b^* a (b^+ a)^* b^+, \\ — R_{1,2}^2 &= R_{1,2}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = b^* a (b^+ a)^*, \\ — R_{1,3}^2 &= R_{1,3}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = b^* a (b^+ a)^* a, \\ — R_{2,1}^2 &= R_{2,1}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = b^+ + b^+ a (b^+ a)^+, \\ — R_{2,2}^2 &= R_{2,2}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = (b^+ a)^*, \\ — R_{2,3}^2 &= R_{2,3}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = b^+ a (\epsilon + (b^+ a)^* a), \\ — R_{3,1}^2 &= R_{3,1}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = b (b^+ a)^* b^+, \\ — R_{3,2}^2 &= R_{3,2}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = b (b^+ a)^*, \\ — R_{3,3}^2 &= R_{3,3}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = \epsilon + a + b (b^+ a)^* a. \end{aligned}$$

Finalement, comme l'état 3 est le seul état final, 1 est l'état initial et qu'il y a 3 états dans l'automate, l'expression régulière associée à cet automate est :

$$\begin{aligned} R_{1,3}^3 &= R_{1,3}^2 + R_{1,3}^2 \cdot (R_{3,3}^2)^* \cdot R_{3,3}^2 \\ &= b^*a(b^+a)^*a + (b^*a(b^+a)^*a)(\epsilon + a + b(b^+a)^*a)^*(\epsilon + a + b(b^+a)^*a) \\ &= (b^*a(b^+a)^*a)(\epsilon + a + b(b^+a)^*a)^*. \end{aligned}$$

3. Pour l'automate dans la figure 10.6c. Nous observons que les états de l'automate sont numérotés à partir de 1. Nous calculons l'expression régulière en utilisant la méthode associant des expressions aux chemins de l'automate. C'est-à-dire, nous calculons $R_{i,j}^k$ pour $k \in [0, 3]$, avec $R_{i,j}^k$ comme définie dans les rappels de cours.

— Pour $k = 0$, nous avons :

$$\begin{array}{lll} — R_{1,1}^0 = \epsilon, & — R_{2,1}^0 = a & — R_{3,1}^0 = b, \\ — R_{1,2}^0 = a, & — R_{2,2}^0 = \epsilon, & — R_{3,2}^0 = a, \\ — R_{1,3}^0 = b, & — R_{2,3}^0 = b, & — R_{3,3}^0 = \epsilon. \end{array}$$

— Pour $k = 1$, nous avons :

$$\begin{aligned} — R_{1,1}^1 &= R_{1,1}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = \epsilon, \\ — R_{1,2}^1 &= R_{1,2}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = a, \\ — R_{1,3}^1 &= R_{1,3}^0 + R_{1,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = b, \\ — R_{2,1}^1 &= R_{2,1}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = a, \\ — R_{2,2}^1 &= R_{2,2}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = \epsilon + aa, \\ — R_{2,3}^1 &= R_{2,3}^0 + R_{2,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = b + ab, \\ — R_{3,1}^1 &= R_{3,1}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,1}^0 = b, \\ — R_{3,2}^1 &= R_{3,2}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,2}^0 = a + ba, \\ — R_{3,3}^1 &= R_{3,3}^0 + R_{3,1}^0 \cdot (R_{1,1}^0)^* \cdot R_{1,3}^0 = \epsilon + bb. \end{aligned}$$

— Pour $k = 2$, nous avons :

$$\begin{aligned} — R_{1,1}^2 &= R_{1,1}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = \epsilon + a(\epsilon + aa)^*a = (aa)^*, \\ — R_{1,2}^2 &= R_{1,2}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = a + a(\epsilon + aa)^*(\epsilon + aa) = a(aa)^*, \\ — R_{1,3}^2 &= R_{1,3}^1 + R_{1,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = b + a(\epsilon + aa)^*(b + ab) = (a + \epsilon)(aa)^*b, \\ — R_{2,1}^2 &= R_{2,1}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = a + (\epsilon + aa)(\epsilon + aa)^*a = (aa)^*a, \\ — R_{2,2}^2 &= R_{2,2}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = \epsilon + aa(\epsilon + aa)^*(\epsilon + aa) = (aa)^*, \\ — R_{2,3}^2 &= R_{2,3}^1 + R_{2,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = b + ab + (\epsilon + aa)(\epsilon + aa)^*(b + ab) = (aa)^*(b + ab), \\ — R_{3,1}^2 &= R_{3,1}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,1}^1 = b + (a + ba)(\epsilon + aa)^*b = b + (a + ba)(aa)^*b, \\ — R_{3,2}^2 &= R_{3,2}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,2}^1 = a + ba + (a + ba)(\epsilon + aa)^*(\epsilon + aa) = (a + ba)aa^*, \\ — R_{3,3}^2 &= R_{3,3}^1 + R_{3,2}^1 \cdot (R_{2,2}^1)^* \cdot R_{2,3}^1 = \epsilon + bb + (a + ba)(\epsilon + aa)^*(b + ab) = \epsilon + bb + (a + ba)(aa)^*(b + ab). \end{aligned}$$

L'expression régulière associée à l'automate est :

$$\begin{aligned} R_{1,3}^3 &= R_{1,3}^2 + R_{1,3}^2 \cdot (R_{3,3}^2)^* \cdot R_{3,3}^2 \\ &= (a + \epsilon)(aa)^*b + ((a + \epsilon)(aa)^*b) \cdot (\epsilon + bb + (a + ba)(aa)^*(b + ab))^* \\ &\quad \cdot (\epsilon + bb + (a + ba)(aa)^*(b + ab)) \\ &= (a + \epsilon)(aa)^*b + ((a + \epsilon)(aa)^*b) \cdot (bb + (a + ba)(aa)^*(b + ab))^* \\ &\quad \cdot (\epsilon bb + (a + ba)(aa)^*(b + ab)). \end{aligned}$$

Solution de l'exercice 121 (page 230)

1. Nous considérons les deux automates l'un après l'autre.

- Considérons l'AFED dans la figure 10.7a.
- Cet automate n'est pas normalisé pour l'initialisation à cause de la transition depuis l'état 0 vers lui-même sur le symbole a et la transition depuis l'état 1 vers l'état 0 sur le symbole a .
- Cet automate n'est pas normalisé pour la terminaison à cause de la transition depuis l'état 1 vers lui-même sur le symbole b et la transition depuis l'état 1 vers l'état 0 sur le symbole a .
- Considérons l' ϵ -AFNED dans la figure 10.7b.
- Cet automate n'est pas normalisé pour l'initialisation à cause de la transition depuis l'état 1 vers l'état 0 sur le symbole a .
- Cet automate n'est pas normalisé pour la terminaison à cause des transitions sortant des états 2 et 3.

Normalisons ces deux automates.

- Pour normaliser pour l'initialisation, nous introduisons un nouvel état qui devient l'état initial et nous ajoutons une ϵ -transition depuis ce nouvel état initial vers l'ancien état initial.
- Pour normaliser pour la terminaison, nous introduisons un nouvel état qui devient l'unique état accepteur et nous ajoutons une ϵ -transition depuis les anciens états accepteurs vers le nouvel état accepteur.

La version normalisée de l'automate dans la figure 10.7a est l'automate dans la figure 10.8a.

La version normalisée de l'automate dans la figure 10.7b est l'automate dans la figure 10.8b.

2. Nous considérons les deux automates l'un après l'autre.

- Nous partons de l'automate dans la figure 10.8a. Nous supprimons les états 1 et 2, dans cet ordre. En supprimant l'état 0 (voir figure 10.9a), nous obtenons l' ϵ -AFEND dans la figure 10.9b. En supprimant l'état 1 (voir figure 10.9c), nous obtenons l' ϵ -AFEND dans la figure 10.9d. Ainsi, l'expression régulière associée à cet automate est celle qui étiquette la transition entre l'état i et l'état f dans l'automate obtenu, c'est-à-dire $(b^* \cdot a) \cdot (b + a \cdot b^* \cdot a)^*$.

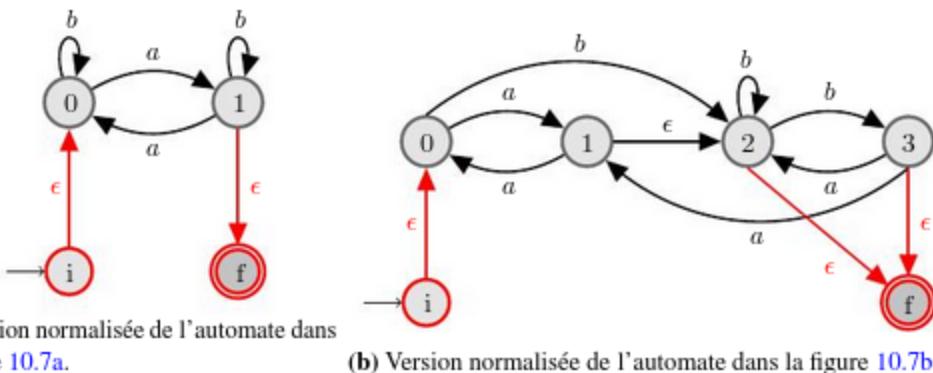


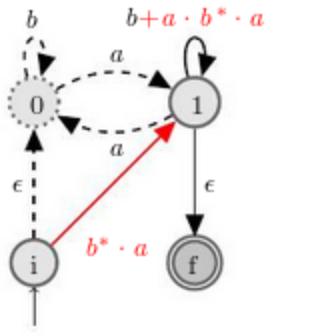
Figure 10.8 – Automates normalisés dans la solution de l'exercice 121.

— Nous partons de l'automate dans la figure 10.8b. Nous supprimons les états 0, 1, 2 et 3, dans cet ordre. En supprimant l'état 0 (voir figure 10.10a), nous obtenons l' ϵ -AFEND dans la figure 10.10b. En supprimant l'état 1 (voir figure 10.10c), nous obtenons l' ϵ -AFEND dans la figure 10.10d. De manière similaire, nous supprimons l'état 2 pour obtenir l' ϵ -AFEND dans la figure 10.10e. Puis, nous supprimons l'état 3, obtenons de manière similaire un automate avec deux états (i et f), et trouvons l'expression régulière suivante :

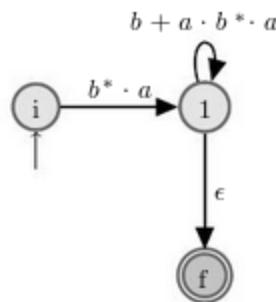
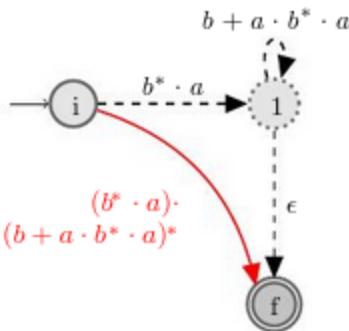
$$\begin{aligned} & b + a(aa)^*(\epsilon + ab)b^* \\ & + \\ & ((b + a(aa)^*(\epsilon + ab))b^+) \cdot ((a(aa)^*(\epsilon + ab))b^+)^* \cdot (\epsilon + (a(aa)^*(\epsilon + ab))b^*) \end{aligned}$$

Solution de l'exercice 122 (page 231)

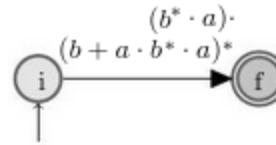
1. Nous avons $EE^*F + F = EE^*F + \epsilon F = (E + \epsilon)E^*F$. En utilisant $(E + \epsilon)E^* = E^*$, nous obtenons $(E + \epsilon)E^*F = E^*F$.
2. Nous prouvons que E^*F est la solution unique de $X = EX + F$.
Supposons que $\epsilon \notin E$ et qu'il existe deux solutions Y et Z telles que $Y \neq Z$. Soit w le plus petit mot tel que $w \in Y$ et $w \notin Z$ (un tel mot existe car $Y \neq Z$). Comme $Z = EZ + F$, nous avons $F \subseteq Z$ et $EZ \subseteq Z$. En conséquence, comme $w \notin Z$, nous avons $w \notin F$. Comme $Y = EY + F$, $w \in Y$, et $w \notin F$, nous avons $w \in EY$. Nous en déduisons l'existence de $w' \in Y$ tel que $w = ew'$ avec $e \neq \epsilon$ (car $\epsilon \notin E$), et donc $|w'| < |w|$. Par ailleurs, comme $w = ew' \notin Z$, nous avons $ew' \notin EZ$. De plus, $w = ew' \notin EZ$ implique que $w' \notin Z$. Finalement, nous avons $w' \in Y$, $w' \notin Z$, and $|w'| < |w|$. Ceci nous fournit une contradiction.



(a) Élimination de l'état 0.

(b) ϵ -AFEND obtenu après élimination de l'état 0.

(c) Élimination de l'état 1.

(d) ϵ -AFEND obtenu après élimination de l'état 1.**Figure 10.9** – Automates pour l'exercice 121 obtenus successivement après élimination des états de l'automate dans la figure 10.8a.

10.5.2 Calcul d'automates à partir d'expressions régulières

Solution de l'exercice 123 (page 231)

Pour rappel, les ϵ -AFEND produits par cette transformation sont normalisés. Nous donnons l'automate généré pour chacun des cas de la définition.

1. Nous séparons les cas de base des cas inductifs.

— Pour les automates de base.

— Pour l'expression régulière ϵ sur l'alphabet Σ , l'automate généré est :

$$(\{i, f\}, \Sigma, i, \{(i, \epsilon, f)\}, \{f\}).$$

— Pour l'expression régulière s sur l'alphabet Σ avec $s \in \Sigma$, l'automate généré est :

$$(\{i, f\}, \Sigma, i, \{(i, s, f)\}, \{f\}).$$

— Pour l'expression régulière \emptyset sur l'alphabet Σ , l'automate généré est :

$$(\{i, f\}, \Sigma, i, \emptyset, \{f\}).$$

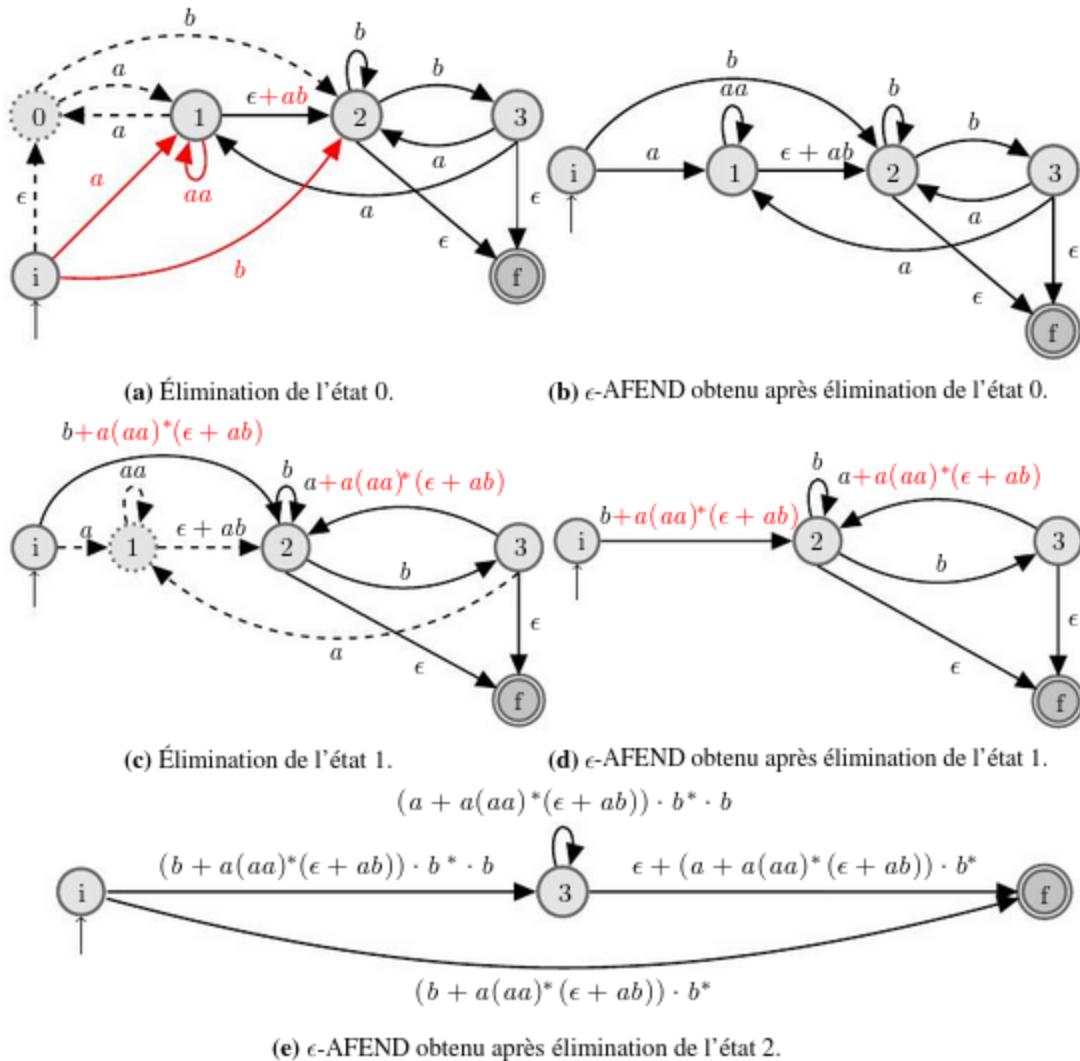


Figure 10.10 – Automates pour l'exercice 121 obtenus successivement après élimination des états de l'automate dans la figure 10.8b.

- Pour les automates composés. Considérons deux automates normalisés $A_1 = (Q_1, \Sigma, q_{\text{init}}^1, \Delta_1, \{f_1\})$ et $A_2 = (Q_2, \Sigma, q_{\text{init}}^2, \Delta_2, \{f_2\})$ reconnaissant les langages dénotés par des expressions régulières e_1 et e_2 respectivement.
- Pour l'expression régulière $e_1 \cdot e_2$, l'automate généré est :

$$(Q_1 \cup Q_2, \Sigma, i_1, \Delta_1 \cup \Delta_2 \cup \{(f_1, \epsilon, i_2)\}, \{f_2\}).$$

L'ensemble d'états de l'automate est l'union des ensembles d'états $(Q_1 \cup Q_2)$. L'état initial est celui du premier automate (i_1). L'état accepteur est celui du second automate (f_2). Les transitions des automates de départ sont

inchangées. Nous ajoutons une transition depuis l'état accepteur du premier automate vers l'état initial du second automate (f_1, ϵ, i_2) .

— Pour l'expression régulière $e_1 + e_2$, l'automate généré est :

$$(Q_1 \cup Q_2 \cup \{i, f\}, \Sigma, i, \Delta_1 \cup \Delta_2 \cup \{(i, \epsilon, i_1), (i, \epsilon, i_2), (f_1, \epsilon, f), (f_2, \epsilon, f)\}, \{f\}).$$

L'ensemble d'états de l'automate est l'union des ensembles d'états $(Q_1 \cup Q_2)$ à laquelle on ajoute deux nouveaux états i et f . L'état initial est l'état i . L'état accepteur est f . Les transitions des automates de départ sont inchangées. Nous ajoutons quatre transitions : depuis l'état initial vers les états initiaux des deux automates $(i, \epsilon, i_1), (i, \epsilon, i_2)$ et depuis les états accepteurs des deux automates vers l'état accepteur $(f_1, \epsilon, f), (f_2, \epsilon, f)$.

— Pour l'expression régulière e_1^* , l'automate généré est :

$$(Q_1 \cup \{i, f\}, \Sigma, i, \Delta_1 \cup \{(i, \epsilon, i_1), (f_1, \epsilon, i_1), (i, \epsilon, f), (f_1, \epsilon, f)\}, \{f\}).$$

L'ensemble d'états est celui de l'automate auquel on ajoute deux nouveaux états i et f . L'état initial est l'état i . L'état accepteur est f . Les transitions de l'automate de départ sont inchangées. Nous ajoutons quatre transitions : depuis l'état initial vers l'état initial de l'automate de départ (i, ϵ, i_1) , depuis l'état initial vers l'état accepteur (i, ϵ, f) , depuis l'état accepteur de l'automate de départ vers l'état initial (f_1, ϵ, i_1) , depuis l'état accepteur de l'automate de départ vers l'état accepteur (f_1, ϵ, f) .

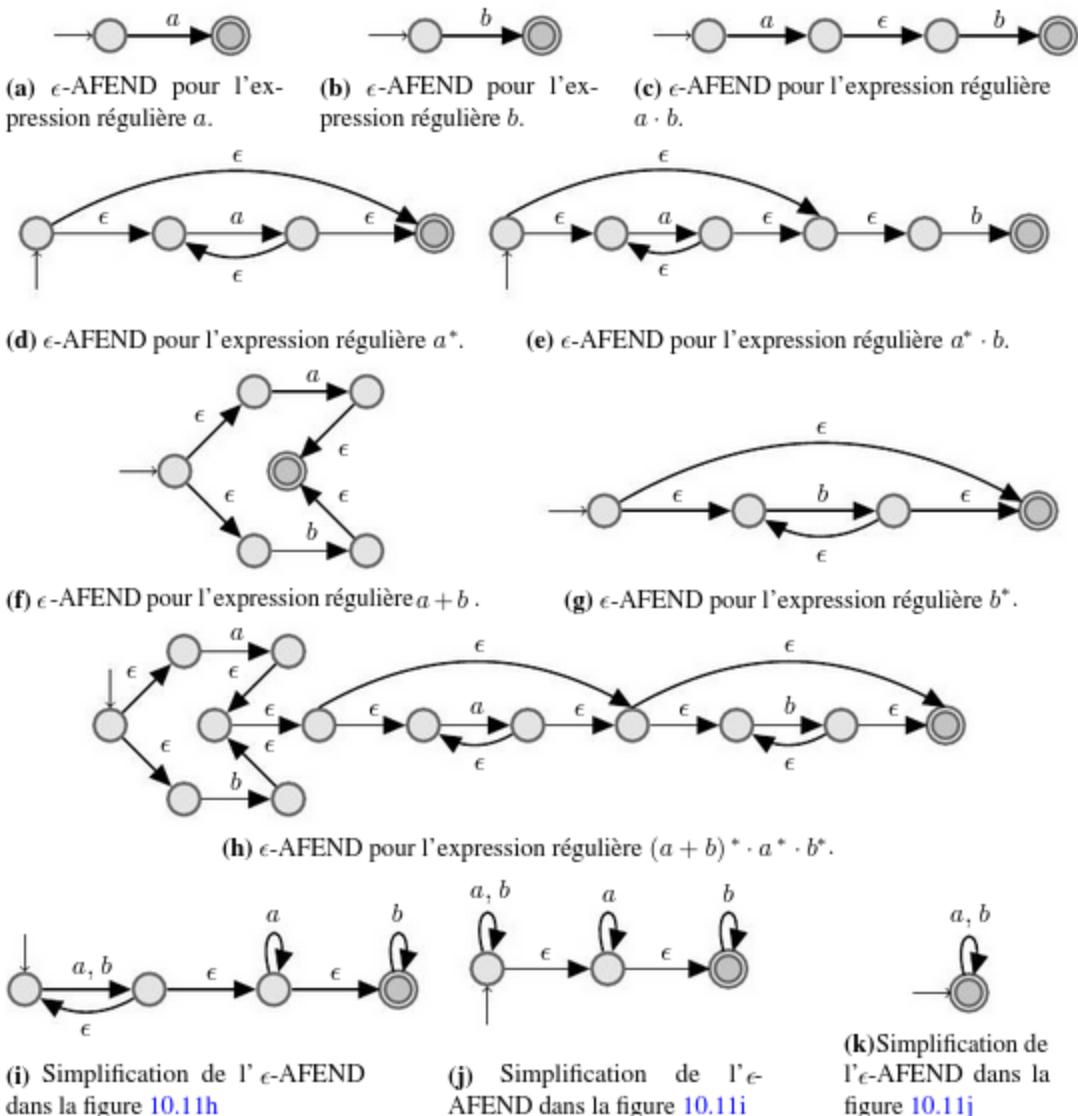
Solution de l'exercice 124 (page 231)

1. Nous partons des ϵ -AFEND associés aux expressions régulières a et b , représentés sur les figures 10.11a et 10.11b, respectivement. Ensuite, pour l'expression régulière $a \cdot b$, nous obtenons l'automate dans la figure 10.11c en utilisant la construction associée à l'opérateur \cdot .
2. Nous procédons de manière similaire. Nous obtenons d'abord l' ϵ -AFEND représenté dans la figure 10.11d correspondant à l'expression régulière a^* (à partir de l'automate associé à l'expression régulière a représenté dans la figure 10.11a). Ensuite, pour l'expression régulière $a^* \cdot b$, nous obtenons l' ϵ -AFEND représenté dans la figure 10.11e en utilisant la construction associée à l'opérateur \cdot et l' ϵ -AFEND associé à l'expression régulière b (figure 10.11b).
3. Nous procédons de manière similaire. Nous obtenons d'abord les ϵ -AFEND correspondant aux expressions régulières $a+b$ et b^* , représentés dans les figures 10.11f et 10.11g, respectivement. Ensuite, nous combinons les ϵ -AFEND en utilisant les constructions pour les opérateurs \cdot et $*$ et obtenons l' ϵ -AFEND représenté dans la figure 10.11h. Nous pouvons aussi simplifier pour obtenir l' ϵ -AFEND équivalent représenté dans la figure 10.11i, puis celui dans la figure 10.11j et enfin celui dans la figure 10.11k.

Solution de l'exercice 125 (page 231)

1. L'automate est représenté dans la figure 10.12a. Nous partons de l'état initial $a \cdot b$ et calculons les successeurs $\frac{\partial}{\partial a}(a \cdot b)$ et $\frac{\partial}{\partial b}(a \cdot b)$. Nous avons :

$$\frac{\partial}{\partial a}(a \cdot b) = (\frac{\partial}{\partial a}a) \cdot b + c(a) \cdot \frac{\partial}{\partial a}b = \epsilon \cdot b + \emptyset \cdot \emptyset = b.$$

Figure 10.11 – ϵ -AFEND obtenus pour l'exercice 124 (p. 231).

$$— \frac{\partial}{\partial b}(a \cdot b) = \left(\frac{\partial}{\partial b}a \right) \cdot b + c(a) \cdot \frac{\partial}{\partial b}b = \emptyset \cdot b + \emptyset \cdot \epsilon = \emptyset.$$

Nous découvrons ainsi deux nouveaux états, b et \emptyset , dont nous calculons les successeurs.

$$— Pour les successeurs de l'état b , nous avons : $\frac{\partial}{\partial a}b = \emptyset$ et $\frac{\partial}{\partial b}b = \epsilon$.$$

$$— Pour les successeurs de l'état \emptyset , nous avons : $\frac{\partial}{\partial a}\emptyset = \frac{\partial}{\partial b}\emptyset = \emptyset$.$$

Nous découvrons ainsi un nouvel état, ϵ , dont nous calculons les successeurs : $\frac{\partial}{\partial a}\epsilon = \frac{\partial}{\partial b}\epsilon = \emptyset$.

De plus, pour déterminer les états accepteurs, nous calculons le terme constant de chacun des états ; les états accepteurs sont ceux qui ont un terme constant égal à ϵ . Nous avons $c(a \cdot b) = c(b) = c(\emptyset) = \emptyset$ et $c(\epsilon) = \epsilon$.

2. L'automate est représenté dans la figure 10.12b. Nous procédons de manière similaire à la question précédente. Nous partons de l'état initial $a^* \cdot b$ et calculons les successeurs :

$$-\frac{\partial}{\partial a}(a^* \cdot b) = \frac{\partial}{\partial a}a^* \cdot b + c(a^*) \cdot \frac{\partial}{\partial a}b = \frac{\partial}{\partial a}a \cdot a^* \cdot b + \epsilon \cdot \emptyset = \epsilon \cdot a^* \cdot b + \emptyset = a^* \cdot b + \emptyset = a^* \cdot b.$$

$$-\frac{\partial}{\partial b}(a^* \cdot b) = \frac{\partial}{\partial b}a^* + c(a^*) \cdot \frac{\partial}{\partial b}b = \frac{\partial}{\partial b}a \cdot a^* \cdot b + \epsilon \cdot \epsilon = \emptyset \cdot a^* \cdot b + \epsilon = \epsilon$$

Nous découvrons ainsi un nouvel état, ϵ , dont nous calculons les successeurs : $\frac{\partial}{\partial a}\epsilon = \frac{\partial}{\partial b}\epsilon = \emptyset$. Nous découvrons ainsi un nouvel état, \emptyset , dont nous calculons les successeurs : $\frac{\partial}{\partial a}\emptyset = \frac{\partial}{\partial b}\emptyset = \emptyset$. Pour déterminer les états accepteurs, nous calculons le terme constant de chacun des états : $c(a^* \cdot b) = c(\emptyset) = \emptyset$ et $c(\epsilon) = \epsilon$.

3. L'automate est représenté dans la figure 10.12c. Nous procédons de manière similaire à la question précédente. Nous partons de l'état initial $(a + b)^* \cdot a$ et calculons les successeurs :

$$-\frac{\partial}{\partial a}(a + b)^* \cdot a = \frac{\partial}{\partial a}(a + b)^* \cdot a + c((a + b)^*) \cdot \frac{\partial}{\partial a}a = \left(\frac{\partial}{\partial a}(a + b) \cdot (a + b)^*\right) \cdot a + \epsilon \cdot \epsilon = (a + b)^* \cdot a + \epsilon, \text{ car } \frac{\partial}{\partial a}(a + b) = \frac{\partial}{\partial a}a + \frac{\partial}{\partial a}b = \epsilon + \emptyset = \epsilon.$$

$$-\frac{\partial}{\partial b}(a + b)^* \cdot a = \frac{\partial}{\partial b}(a + b)^* \cdot a + c((a + b)^*) \cdot \frac{\partial}{\partial b}a = \left(\frac{\partial}{\partial b}(a + b) \cdot (a + b)^*\right) \cdot a + \epsilon \cdot \emptyset = (a + b)^* \cdot a, \text{ car } \frac{\partial}{\partial b}(a + b) = \frac{\partial}{\partial b}a + \frac{\partial}{\partial b}b = \emptyset + \epsilon = \epsilon.$$

De plus, pour déterminer les états accepteurs, nous avons : $c((a + b)^* \cdot a) = \emptyset$ et $c((a + b)^* \cdot a + \epsilon) = \epsilon$.

4. L'automate est représenté dans la figure 10.12d. Nous procédons de manière similaire à la question précédente en calculant les dérivées des états découverts, en commençant à dériver à partir de l'état initial qui est l'expression régulière initiale. Les états accepteurs sont ceux dont l'expression régulière a un terme constant égal à ϵ .

Solution de l'exercice 126 (page 232)

1. L'automate est représenté dans la figure 10.13a.
2. L'automate est représenté dans la figure 10.13b.

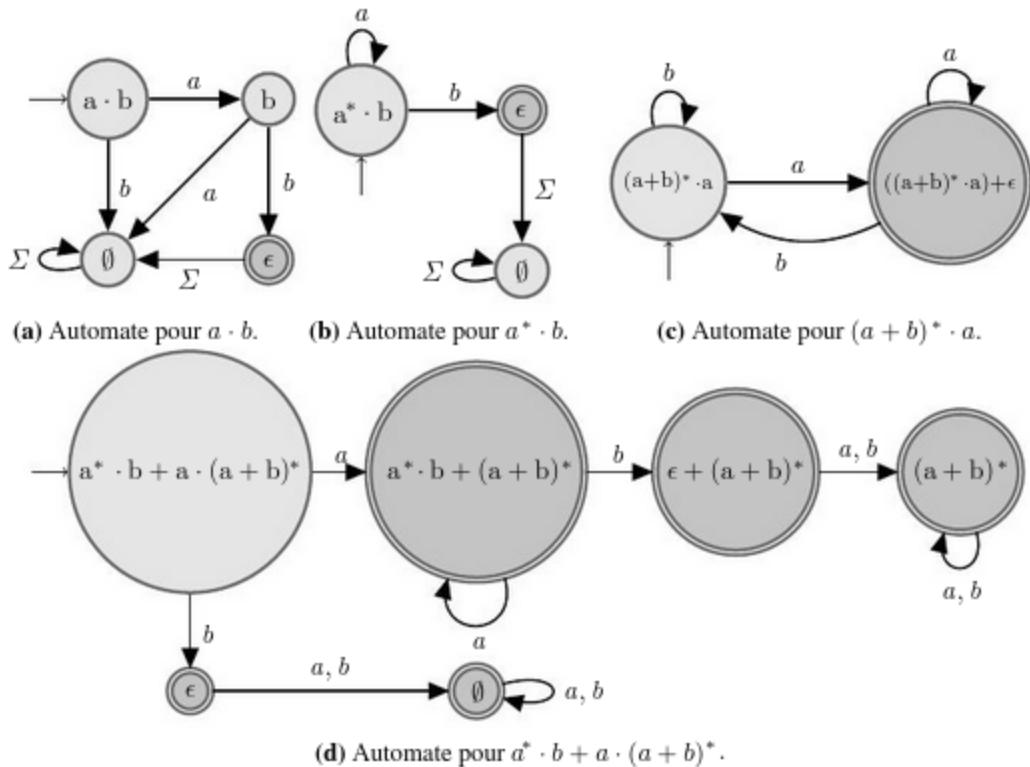


Figure 10.12 – Automates pour les expressions régulières de l'exercice 125 (p. 231).

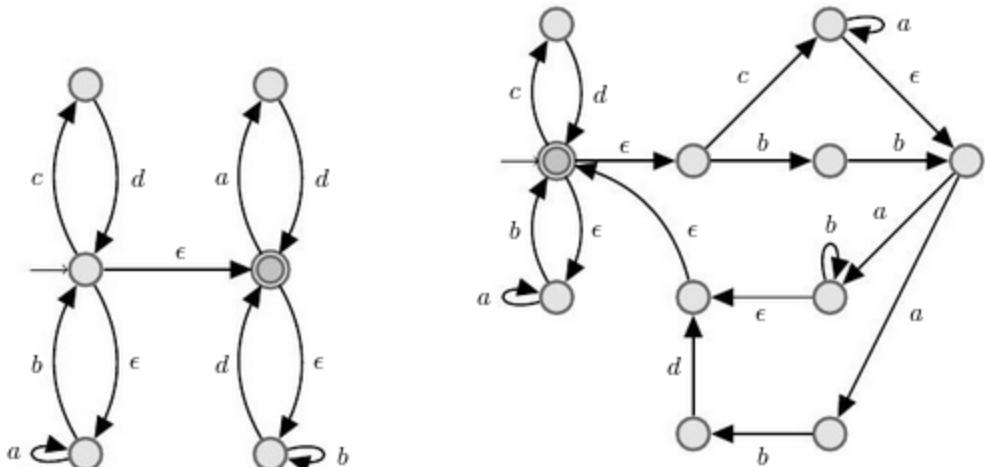


Figure 10.13 – ϵ -AFEND obtenus pour l'exercice 126 (p. 232).

Grammaires

11.1 Résumé intuitif du chapitre

Nous abordons dans ce chapitre la notion de **grammaire**, notion qui est étroitement associée à un langage informatique ou à une langue naturelle, comme le Français etc. Une grammaire permet de générer des éléments du langage. Une grammaire est caractérisée par quatre éléments : un **vocabulaire terminal**, un **vocabulaire non terminal**, un **axiome** (qui est un élément du vocabulaire non terminal) et un ensemble de **règles de production**. Si Σ est le vocabulaire terminal, on s'intéresse aux éléments de Σ^* que l'on peut générer à partir de l'axiome en appliquant les règles de production. On rappelle aussi la **classification de Chomsky** des grammaires, qui les réparti en quatre familles, les **grammaires régulières** (linéaires à droite ou à gauche), les grammaire algébriques ou hors contextes, les grammaires sous contexte et les grammaires de type général.

Dans le chapitre suivant, nous ferons le lien entre grammaire régulière, expressions régulières et automates. En pratique, les grammaires régulières peuvent être utilisées pour caractériser le lexique d'un langage informatique tandis que les grammaires hors contextes sont utilisées en analyse syntaxique pour les langages informatiques.

11.2 Les notions essentielles

11.2.1 Définition des grammaires

Définition 147 (Grammaire) *Une grammaire est un quadruplet (V_T, V_N, Z, P) tel que V_T et V_N sont disjoints ($V_T \cap V_N = \emptyset$) et :*

- *V_T est un ensemble fini appelé le vocabulaire terminal qui est l'ensemble des symboles terminaux,*
- *V_N est un ensemble fini appelé le vocabulaire non terminal qui est l'ensemble des symboles non terminaux,*
- *$Z \in V_N$ est l'axiome et*

- $P \subseteq (V_T \cup V_N)^* \times (V_T \cup V_N)^*$ est l'ensemble des règles de production.

Définition 148 (Vocabulaire d'une grammaire) Le vocabulaire d'une grammaire (V_T, V_N, Z, P) est l'ensemble $V = V_T \cup V_N$.

On note $x \rightarrow y$ pour $(x, y) \in P$.

11.2.2 Dérivations en une et plusieurs étapes

Dans la suite, nous considérons une grammaire $G = (V_T, V_N, Z, P)$ et son vocabulaire $V = V_T \cup V_N$.

Définition 149 (Relation de dérivation en une étape) G permet de dériver un mot $u \in V^*$ en un mot $v \in V^*$ en une étape, noté $u \Rightarrow v$, s'il existe une règle $x \rightarrow y \in P$ telle que :

- $u = \alpha \cdot x \cdot \beta$, et
- $v = \alpha \cdot y \cdot \beta$,

avec $\alpha, \beta \in V^*$. Les mots u et v sont appelés formes sententielles.

Définition 150 (Relation de dérivation en plusieurs étapes) La relation de dérivation en plusieurs étapes, notée \Rightarrow^* , est la fermeture transitive de \Rightarrow .

11.2.3 Langage généré par une grammaire

Si $Z \Rightarrow^* w$ et $w \in V_T^*$, alors on dit que le mot w est généré par la grammaire.

Définition 151 (Langage généré par une grammaire) Le langage engendré par une grammaire, noté $\mathcal{L}_{\text{gram}}(G)$, est défini par : $\mathcal{L}_{\text{gram}}(G) = \{w \in V_T^* \mid Z \Rightarrow^* w\}$.

11.2.4 Classification de Chomsky des grammaires

Les contraintes sur la forme de la grammaire définissent son type.

Définition 152 (Classification de Chomsky des grammaires) La classification de Chomsky des grammaires est décrite dans le tableau 11.1. Cette classification regroupe les grammaires par type selon des contraintes sur les règles de production utilisées dans la définition des grammaires. Les différents types de grammaires reconnaissent différentes classes de langages. Par exemple, les grammaires de type 2 sont dites non contextuelles et reconnaissent les langages dits hors contextes ou encore appelés algébriques. Les grammaires de type 3 sont dites régulières ou encore appelées rationnelles et peuvent prendre deux formes, linéaires à droite ou linéaires à gauche. Elles génèrent les langages réguliers, comme nous le verrons dans le chapitre 12.

Cette classification est hiérarchique : pour $i, j \in [0, 3]$, avec $i \leq j$, les grammaires de type i sont plus générales que les grammaires de type j ; autrement dit, l'ensemble des grammaires de type i contient l'ensemble des grammaires de type j . On peut le voir sur les contraintes associées aux règles : pour i et j comme ci-dessus, la contrainte associée aux grammaires de type j est plus forte que la contrainte associée aux grammaires de type i .

Tableau 11.1 – Description de la classification de Chomsky des grammaires. Pour rappel, le symbole \cdot dénote l'opération de concaténation et le symbole $*$ dénote la fermeture de Kleene d'un ensemble. Les ensembles V, V_N et V_T sont des vocabulaires. Ainsi, par exemple, $x \in V^* \cdot V_N \cdot V^*$ indique que x peut s'écrire $x_1 \cdot x_2 \cdot x_3$ avec $x_1 \in V^*, x_2 \in V_N$ et $x_3 \in V^*$.

Type	Appellation	Contrainte(s)
0	générale	$x \rightarrow y$ avec $x \in V^* \cdot V_N \cdot V^*$ et $y \in V^*$
1	contextuelle	$\alpha \cdot A \cdot \beta \rightarrow \alpha \cdot \gamma \cdot \beta$ avec $\alpha, \beta, \gamma \in V^*, \gamma \neq \epsilon$ et $A \in V_N$
2	non contextuelle	$A \rightarrow \gamma$ avec $A \in V_N$ et $\gamma \in V^*$
3	régulière	linéaire gauche $A \rightarrow B \cdot a \mid A \rightarrow a \mid A \rightarrow \epsilon$ linéaire droite $A \rightarrow a \cdot B \mid A \rightarrow a \mid A \rightarrow \epsilon$ avec $A, B \in V_N$ et $a \in V_T$

De plus, on distingue parfois les grammaires dites linéaires qui sont telles que chaque règle de production possède exactement un symbole non terminal en partie gauche et au plus un symbole non terminal dans sa partie droite (il n'y a pas de contrainte sur la position de ce symbole non terminal). C'est-à-dire que les règles sont de la forme $A \rightarrow \alpha \cdot B \cdot \beta$ avec $\alpha, \beta \in V_T^*$. Par exemple, la grammaire contenant la règle de production $A \rightarrow aAa$ est linéaire, mais n'est pas linéaire à droite ni à gauche.

Notre étude principale se fera sur les grammaires régulières et leur lien avec les langages réguliers et les automates, ce que nous ferons au chapitre 12.

11.3 Exercices

11.3.1 Générer des mots avec les grammaires

Exercice 127 (♠) — Dérivations

Considérons la grammaire $G = (\{a, b, c\}, \{Z\}, Z, P)$ où P est donné par les règles de production suivantes : $Z \rightarrow abZ \mid bcZ \mid bbZ \mid a \mid cb$. Donner une dérivation en plusieurs étapes de G pour les mots suivants lorsque cela est possible.

1. $bcbba$.
2. $bbbcbba$.
3. bbb .
4. $bcabbbbcbb$.

Exercice 128 (♠) — Mots générés par une grammaire

Considérons la grammaire $G = (V_T, V_{NT}, PH, P)$, où :

— V_T est l'ensemble des symboles terminaux suivants :

$\{un, une, l', etudiante, etudiant, enseignante, enseignant, dutennis, duski, descours, faitdu, etudie, donne\}$

— P est l'ensemble des règles de production ¹ suivantes :

$$\begin{array}{ll} PH \rightarrow GN\,GV & GN \rightarrow AF\,NF \mid AM\,NM \\ GV \rightarrow V\,C & NF \rightarrow etudiante \mid enseignante \\ AF \rightarrow une \mid l' & AM \rightarrow un \mid l' \\ V \rightarrow fait \mid etudie \mid donne & C \rightarrow dutennis \mid duski \mid descours \\ NM \rightarrow etudiant \mid enseignant & \end{array}$$

1. Donner quelques mots générés par la grammaire (pseudo phrases en français).
2. Donner quelques phrases non générées par la grammaire.
3. Donner le nombre de phrases générées par cette grammaire. Il n'est pas requis que les phrases aient du sens.

11.3.2 Langages et grammaires

Exercice 129 (♦) — Langage généré par une grammaire

Indiquer les langages générés par les grammaires de la forme $(\{a, b\}, \{Z, T, U\}, Z, P)$ avec les ensembles de règles de production suivants P :

$$1. \left\{ \begin{array}{l} Z \rightarrow aU, \\ U \rightarrow bU \mid aT \mid \epsilon, \\ T \rightarrow aU \mid bT \end{array} \right\}. \quad 2. \left\{ \begin{array}{l} Z \rightarrow aT \mid bZ, \\ T \rightarrow aT \mid \epsilon \end{array} \right\}.$$

Exercice 130 (♦♦) — Langage généré par une grammaire

Indiquer les langages générés par les grammaires de la forme $(\{a, b\}, \{Z, A, B\}, Z, P)$ avec les ensembles de règles de production suivants P :

$$\begin{array}{ll} 1. \left\{ \begin{array}{l} Z \rightarrow AB, \\ A \rightarrow ab, \\ B \rightarrow BB \end{array} \right\}. & 4. \left\{ \begin{array}{l} Z \rightarrow AA \mid B, \\ A \rightarrow aaA \mid aa, \\ B \rightarrow bB \mid B \end{array} \right\}. \\ 2. \left\{ \begin{array}{l} Z \rightarrow AB \mid aA, \\ A \rightarrow a, \\ B \rightarrow b \end{array} \right\}. & 5. \left\{ \begin{array}{l} Z \rightarrow AB \mid aAb, \\ B \rightarrow bBa \mid \epsilon, \\ A \rightarrow \epsilon \end{array} \right\}. \\ 3. \left\{ \begin{array}{l} Z \rightarrow AB \mid AA, \\ A \rightarrow aB, \\ B \rightarrow b \end{array} \right\}. & \end{array}$$

1. Dans les règles de production PH est l'abréviation de phrase, V de verbe, AF d'article féminin, GN de groupe nominal, NF de nom féminin, AM d'article masculin, GV de groupe verbal, NM de nom masculin et C de complément.

Exercice 131 (♠♠) — Grammaire qui génère un langage

Pour chacun des langages suivants, donner une grammaire qui le génère et générer quelques mots. Indiquer si cette grammaire est linéaire à droite ou linéaire à gauche.

- | | |
|----------------------------------------------------------|----------------------------------------------------------|
| 1. $\{a^n \cdot b^n \mid n \in \mathbb{N}\}.$ | 4. $\{b \cdot a^n \cdot b \mid n \in \mathbb{N}\}.$ |
| 2. $\{a^n \cdot b^{2 \times n} \mid n \in \mathbb{N}\}.$ | 5. $\{a^n \cdot b \cdot c^m \mid n, m \in \mathbb{N}\}.$ |
| 3. $\{a^n \cdot b^m \mid n, m \in \mathbb{N}\}.$ | 6. $\{a^{n+1} \cdot b^n \mid n \in \mathbb{N}\}.$ |

Exercice 132 (♠♠) — Grammaire générant les chaînes de bits

Considérons l'alphabet $\Sigma = \{0, 1\}$ contenant les symboles utilisés dans la représentation binaire des entiers naturels. Nous considérons le langage formé par les entiers naturels pairs écrits en binaire et dans la convention gros-boutiste : les mots dans ce langage terminent par un 0 et ne commencent pas par un 0 sauf pour le mot représentant l'entier naturel 0.

1. Donner une définition formelle de ce langage.
2. Écrire une grammaire linéaire à droite qui génère ce langage.
3. Écrire une grammaire linéaire à gauche qui génère ce langage.

Exercice 133 (♠♠♠) — Composition de grammaires

Soient L et L' des langages réguliers générés par les grammaires G et G' respectivement. On souhaite démontrer qu'il existe des grammaires régulières qui génèrent les langages suivants : $L \cup L'$, $L \cdot L'$, $(L)^*$.

1. À partir d'exemples de grammaires, montrer comment construire ces grammaires.
2. Généraliser. C'est-à-dire, donner les grammaires pour les langages demandés à partir de grammaires régulières quelconques. Expliquer la construction de ces grammaires.

11.4 Indications pour résoudre les exercices

Indications pour l'exercice 133 (p. 251)

1. *Les modifications se font en introduisant un nouveau symbole non terminal qui devient l'axiome et est relié aux anciens axiomes par des règles de production.*
2. *Considérer deux grammaires (T, N, Z, P) et (T', N', Z', P') pour les langages L et L' . Supposer que les ensembles N et N' sont disjoints, ce qui est toujours possible après un renommage des symboles non terminaux. Suivre le principe utilisé pour la question précédente.*

11.5 Solutions des exercices

11.5.1 Générer des mots avec les grammaires

Solution de l'exercice 127 (page 249)

- Pour cette grammaire, nous pouvons obtenir la dérivation : $Z \Rightarrow bcZ \Rightarrow bccbZ \Rightarrow bcbba$
- Pour cette grammaire, nous pouvons obtenir la dérivation : $Z \Rightarrow bbZ \Rightarrow bbbcZ \Rightarrow bbbccbZ \Rightarrow bbbcbba$.
- Pour cette grammaire, on remarque que l'application de n successives génère des mots de longueur supérieure ou égale à n . Pour le mot de longueur 3 bbb , on constate qu'en appliquant 3 dérivation successives, le mot bbb n'appartient aux mots engendrés.
- De manière similaire, pour cette grammaire, aucune des dérивations possibles ne permet de générer le mot $bcabbbbcb$. Pour montrer cela, il faut construire l'ensemble des dérivations dont la longueur est inférieure ou égale à la longueur du mot en question.

Solution de l'exercice 128 (page 249)

- Nous donnons quelques mots générés par la grammaire (pseudo phrases en français). Nous utilisons le symbole de la concaténation (\cdot) pour la lisibilité.
 - $un \cdot etudiant \cdot etudie \cdot descours$
 - $l' \cdot enseignante \cdot fait \cdot dutennis$
- Nous donnons quelques mots qui ne peuvent être générés par la grammaire (pseudo phrases en français). Nous utilisons le symbole de la concaténation (\cdot) pour la lisibilité.
 - $une \cdot etudiant \cdot etudie \cdot descours$
 - $un \cdot enseignante \cdot fait$
- Il y a $|GN| \times |GV|$ phrases générées par cette grammaire, où $|GN|$ et $|GV|$ sont les cardinaux des ensembles de mots générés à partir des symboles non terminaux GN et GV . Pour $|GN|$, suivant un principe similaire, nous trouvons $|GN| = |AF| \times |NF| + |AM| \times |NM|$, avec $|AF| = |NF| = |NM| = |AM| = 2$. Pour $|GV|$, suivant un principe similaire, nous trouvons $|GV| = |V| \times |C|$, avec $|V| = |C| = 3$. Au final, il y a 72 mots générés par cette grammaire.

11.5.2 Langages et grammaires

Solution de l'exercice 129 (page 250)

- Le langage généré par cette grammaire est l'ensemble des mots sur $\{a, b\}$ contenant un nombre impair de a .
- Le langage généré par cette grammaire est :

$$\{b^i a^j \mid i, j \in \mathbb{N} \wedge j > 0\}.$$

Solution de l'exercice 130 (page 250)

- Le langage généré par cette grammaire est vide. En effet, la dérivation de l'axiome S force à générer le symbole non terminal B qui ne permet pas de terminer la dérivation et d'obtenir des symboles non terminaux.
- Le langage généré par cette grammaire est : $\{a^j \mid j \in \mathbb{N} \setminus \{0\}\} \cup \{a^j \cdot b \mid j \in \mathbb{N}\}$. Ce langage peut être décrit par l'expression régulière $a^+(b + \epsilon)$.
- Le langage généré par cette grammaire est : $\{ab\} \cup \{abab\} = \{ab, abab\}$.
- Le langage généré par cette grammaire est : $\{a^i \mid i \in \mathbb{N} \wedge i \bmod 2 = 0 \wedge i \geq 4\}$. Nous remarquons en particulier que le symbole non terminal B ne permet pas de générer de mots.
- Le langage généré par cette grammaire est : $\{ab\} \cup \{b^i a^i \mid i \in \mathbb{N}\}$.

Solution de l'exercice 131 (page 251)

Nous utilisons la convention suivante. Nous donnons directement l'ensemble des règles de production de la grammaire; ceci définit les symboles non terminaux et les symboles terminaux de la grammaire, ainsi que l'axiome qui est le symbole non terminal utilisé dans la première règle de dérivation. Ainsi, dans les ensembles de règles de production suivants les symboles Z, A, B sont des symboles non terminaux, les symboles a, b et c sont des terminaux, le symbole Z est l'axiome.

Pour certains langages, nous proposons plusieurs solutions.

- Nous proposons les trois grammaires équivalentes suivantes :

- $\{Z \rightarrow aZb \mid \epsilon\}$;
- $\{Z \rightarrow AZB \mid \epsilon, A \rightarrow a, B \rightarrow b\}$;
- $\{Z \rightarrow AB \mid AZB, A \rightarrow a \mid \epsilon, B \rightarrow b \mid \epsilon\}$.

Ces grammaires ne sont pas régulières (donc ni linéaire à droite ni à gauche).

- Nous proposons les trois grammaires équivalentes suivantes :

- $\{Z \rightarrow aSbb \mid \epsilon\}$;
- $\{Z \rightarrow AZB \mid \epsilon, A \rightarrow a, B \rightarrow bb\}$;
- $\{Z \rightarrow ASBB \mid \epsilon, A \rightarrow a, B \rightarrow b\}$.

Ces grammaires ne sont pas régulières.

- Nous proposons les trois grammaires équivalentes suivantes :

- $\{Z \rightarrow aA \mid bB \mid \epsilon, A \rightarrow aA \mid bB \mid \epsilon, B \rightarrow bB \mid \epsilon\}$, cette grammaire est régulière et linéaire à droite;
- $\{Z \rightarrow AB, A \rightarrow aA \mid \epsilon, B \rightarrow Bb \mid \epsilon\}$, cette grammaire n'est pas régulière;
- $\{Z \rightarrow AZ \mid ZB \mid \epsilon, A \rightarrow a, B \rightarrow b\}$, cette grammaire n'est pas régulière.

- Nous proposons les deux grammaires équivalentes suivantes :

- $\{Z \rightarrow bAb, A \rightarrow AA \mid a\}$;
- $\{Z \rightarrow bAb, A \rightarrow aA \mid \epsilon\}$.

Ces deux grammaires ne sont pas régulières.

5. Nous proposons la grammaire $\{Z \rightarrow b \mid abc \mid aZc\}$ qui n'est pas régulière.
6. Nous proposons la grammaire $\{Z \rightarrow aS, S \rightarrow \epsilon \mid aSb\}$ qui n'est pas régulière.

Remarque 47 Dans les solutions suivantes, nous représentons également les grammaires par leurs ensembles de règles de production, en utilisant la convention décrite dans l'exercice 131.

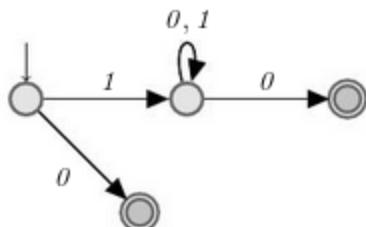
Solution de l'exercice 132 (page 251)

1. Nous pouvons définir l'ensemble en intension :

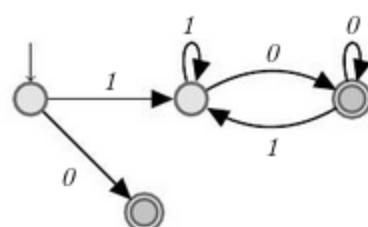
$$\{0\} \cup \{1 \cdot u_0 \cdots u_n \cdot 0 \mid \forall i \in [0, n]. u_i \in \{0, 1\}\}.$$

Nous pouvons également définir le langage comme celui dénoté par l'expression régulière $0 + 1 \cdot \Sigma^* \cdot 0$ sur l'alphabet $\Sigma = \{0, 1\}$. Nous pouvons enfin définir le langage comme celui reconnu par l'AFEND représenté dans la figure 11.1a et l'AFED équivalent représenté dans la figure 11.1b.

2. Une grammaire linéaire à droite qui génère ce langage est donnée dans la figure 11.2a.
3. Une grammaire linéaire à gauche qui génère ce langage est donnée dans la figure 11.2b.



(a) Version AFEND.



(b) Version AFED.

Figure 11.1 – Automates solutions de la première question de l'exercice 132 reconnaissant les entiers naturels écrits en binaire (sur l'alphabet $\{0, 1\}$) en convention gros-boutiste.

$$\left\{ \begin{array}{l} S \rightarrow 0 \mid 1 \cdot U, \\ U \rightarrow 0 \cdot U \mid 1 \cdot U \mid 0 \end{array} \right\}.$$

(a) Version linéaire à droite.

$$\left\{ \begin{array}{l} S \rightarrow 0 \mid U, \\ U \rightarrow U \cdot 0 \mid U \cdot 1 \mid 1 \end{array} \right\}.$$

(b) Version linéaire à gauche.

Figure 11.2 – Grammaires solutions des questions 2. et 3. de l'exercice 132 générant les entiers naturels écrits en binaire (sur l'alphabet $\{0, 1\}$) en convention gros-boutiste.

Solution de l'exercice 133 (page 251)

1. Considérons les deux grammaires dans les figures 11.3a et 11.3b reconnaissant des langages L et L' , respectivement.

$\{ S \rightarrow \epsilon \mid aSb \}$	$\left\{ \begin{array}{l} Z \rightarrow aT \mid aZb, \\ T \rightarrow aT \mid bU, \\ U \rightarrow bU \mid \epsilon \end{array} \right\}$
(a) Un exemple de grammaire L .	(b) Un exemple de grammaire L' .
$\left\{ \begin{array}{l} Y \rightarrow S \mid Z, \\ S \rightarrow \epsilon \mid aSb, \\ Z \rightarrow aT \mid aZb, \\ T \rightarrow aT \mid bU, \\ U \rightarrow bU \mid \epsilon \end{array} \right\}$	$\left\{ \begin{array}{l} Y \rightarrow S \cdot Z, \\ S \rightarrow \epsilon \mid aSb, \\ Z \rightarrow aT \mid aZb, \\ T \rightarrow aT \mid bU, \\ U \rightarrow bU \mid \epsilon \end{array} \right\}$
(c) Grammaire générant $L \cup L'$.	(d) Grammaire générant $L \cdot L'$.
$\left\{ \begin{array}{l} R \rightarrow R \cdot S \mid \epsilon, \\ S \rightarrow \epsilon \mid aSb \end{array} \right\}$	$\left\{ \begin{array}{l} Y \rightarrow Y \cdot Z \mid \epsilon, \\ Z \rightarrow aT \mid aZb, \\ T \rightarrow aT \mid bU, \\ U \rightarrow bU \mid \epsilon \end{array} \right\}$
(e) Grammaire générant L^* .	(f) Grammaire générant L'^* .

Figure 11.3 – Grammaires pour l'exercice 133.

- La grammaire générant $L \cup L'$ est donnée dans la figure 11.3c. Nous avons pris l'ensemble des règles des deux grammaires de départ et introduit la règle $Y \rightarrow S \mid Z$ qui permet après réécriture de l'axiome de choisir entre la génération d'un mot par le symbole non terminal S ou Z qui étaient les axiomes des grammaires de départ.
 - La grammaire générant $L \cdot L'$ est donnée dans la figure 11.3d. Nous avons pris l'ensemble des règles des deux grammaires de départ et introduit la règle $Y \rightarrow S \cdot Z$ qui permet après réécriture de l'axiome de former un mot qui est la concaténation de deux mots générés par les symboles non terminaux S et Z qui étaient les axiomes des grammaires de départ.
 - La grammaire reconnaissant L^* est donnée dans la figure 11.3e. Nous avons pris l'ensemble des règles de la grammaire de départ et introduit la règle $R \rightarrow R \cdot S \mid \epsilon$ qui permet de former un mot par un nombre arbitraire de concaténations de mots générés à partir du symbole non terminal S qui était l'axiome de la grammaire de départ.
 - La grammaire générant L'^* est donnée dans la figure 11.3f. Le principe est similaire avec ici l'introduction de la règle $Y \rightarrow Y \cdot Z \mid \epsilon$.
2. Nous généralisons le principe utilisé à la question précédente. Soient (T, N, Z, P) et (T', N', Z', P') les grammaires pour L et L' . Nous supposons que les ensembles N et N' sont disjoints, ce qui est toujours possible après un renommage des symboles non terminaux. Soit $Z'' \notin N \cup N'$, un nouveau symbole non terminal.
- La grammaire reconnaissant $L \cup L'$ est donnée ci-dessous :

$$(T \cup T', N \cup N' \cup \{Z''\}, Z'', P \cup P' \cup \{Z'' \rightarrow Z \mid Z'\}).$$

Nous avons considéré l'union des symboles non terminaux, terminaux et règles de production. Nous avons également ajouté le symbole non terminal Z'' qui devient l'axiome de la nouvelle grammaire et la règle de production $Z'' \rightarrow Z \mid Z'$ introduisant le choix entre la génération d'un mot généré par l'une des grammaires.

- La grammaire reconnaissant $L \cdot L'$ est donnée ci-dessous :

$$(T \cup T', N \cup N' \cup \{Z''\}, Z'', P \cup P' \cup \{Z'' \rightarrow Z \cdot Z'\}).$$

De manière similaire, nous avons considéré l'union des symboles non terminaux, terminaux et règles de production. Nous avons également ajouté le symbole non terminal Z'' qui devient l'axiome de la nouvelle grammaire et la règle de production $Z'' \rightarrow Z \cdot Z'$ permettant la génération d'un mot formé par la concaténation de mots générés par les deux grammaires.

- La grammaire reconnaissant L^* est donnée ci-dessous :

$$(T, N \cup \{Z'\}, Z', P \cup \{Z' \rightarrow Z' \cdot Z \mid \epsilon\}).$$

Nous avons considéré les symboles terminaux, symboles non terminaux et règles de production de la grammaire de départ. Nous avons également ajouté le symbole non terminal Z' qui devient l'axiome de la nouvelle grammaire et qui permet de générer des mots formés par concaténation de mots générés par Z .

Notons que la grammaire $(T, N \cup \{Z'\}, Z', P \cup \{Z' \rightarrow Z \cdot Z' \mid \epsilon\})$ est équivalente ; les mots de L sont générés « par la gauche » des règles.

Grammaires régulières

12.1 Résumé intuitif du chapitre

Dans ce chapitre, nous reprenons la notion de grammaire régulière qui sont caractérisées par le fait que leur production sont de la forme, pour une grammaire linéaire droite, $A \rightarrow aB$, $A \rightarrow \epsilon$, $A \rightarrow a$ où A, B sont des éléments non terminaux, a un élément terminal. À ce type de grammaire, on peut associer un automate. Les états de l'automate sont les éléments du vocabulaire terminal augmentés d'un état accepteur F . Aux productions $A \rightarrow aB$, $A \rightarrow a$ et $A \rightarrow \epsilon$, on associe respectivement les transitions (A, a, B) , (A, a, F) et (A, ϵ, F) . On peut aussi, mais nous ne développons aucun exercice sur ce thème, associé un automate à une grammaire linéaire à gauche. Réciproquement, on peut associer une grammaire à un automate.

On voit ainsi qu'un langage à états peut être indifféremment caractérisé par un automate, une grammaire régulière ou une expression régulière.

12.2 Les notions essentielles

12.2.1 Grammaires et automates

Nous nous intéressons plus particulièrement au lien entre grammaires régulières et automates.

Remarque 48 (Affaiblissement des contraintes sur les grammaires linéaires droites) *Pour simplifier l'écriture des grammaires, nous nous autorisons d'affaiblir les contraintes sur les grammaires régulières. De telles règles peuvent être transformées en règles respectant les contraintes des grammaires linéaires droites, sans changer le langage généré par la grammaire. Les facilités de notation ci-dessous sont données pour les grammaires linéaires droites, mais s'appliquent également aux grammaires linéaires gauches.*

- *Nous autorisons les règles de la forme $A \rightarrow u \cdot B$ où A et B sont des symboles non terminaux et $u = u_1 \dots u_n$ un mot sur $\Sigma^* \setminus \{\epsilon\}$ (avec $u_i \in \Sigma$). Pour satisfaire les*

contraintes des grammaires linéaires droite, on transforme chaque règle $A \rightarrow u \cdot B$ en :

$$\begin{array}{lcl} A & \rightarrow & u_1 \cdot A _u_1_B, \\ A_u_1_B & \rightarrow & u_2 \cdot A _u_2_B, \\ & \cdots & \\ A_u_i_B & \rightarrow & u_{i+1} \cdot A _u_{i+1}_B, \\ & \cdots & \\ A_u_{n-1}_B & \rightarrow & u_n \cdot B. \end{array}$$

c'est-à-dire, nous introduisons les symboles non terminaux $A_u_1_B, \dots, A_u_n_B$ intermédiaires et introduisons les règles $A \rightarrow u_1 \cdot A_u_1_B$ permettant de générer u_1 et $A_u_i_B \rightarrow u_{i+1} \cdot A_u_{i+1}_B$ permettant de générer le symbole u_{i+1} , pour $i \in [1, n - 1]$.

- *Nous autorisons les règles de la forme $A \rightarrow B$ où A et B sont des symboles non terminaux. Pour satisfaire les contraintes des grammaires linéaires droites, nous remplaçons la règle $A \rightarrow B$ par l'ensemble de règles $\{A \rightarrow y \mid B \rightarrow y\}$.*

Dans le cas des grammaires régulières, l'ensemble des symboles non terminaux « correspond » à l'alphabet des automates :

- Un automate reconnaît des mots sur son alphabet ¹.
- Une grammaire génère des mots sur son ensemble de symboles terminaux.

Théorème 14 (Équivalence entre grammaires régulières et AFED) *Nous avons l'équivalence suivante entre grammaires régulières et AFED.*

1. *À chaque grammaire régulière, on peut associer un ϵ -AFEND qui reconnaît le même langage.*
2. *Réciproquement, à chaque AFED, on peut associer une grammaire régulière qui reconnaît le même langage.*

Ces deux points se démontrent en donnant des procédures de traduction entre grammaires et automates.

Corollaire 7 *Les grammaires régulières génèrent les langages à états (c'est-à-dire les langages réguliers par le théorème de Kleene).*

Corollaire 8 (Retour sur la fermeture des langages réguliers) *Étant données deux grammaires G et G' générant des langages réguliers L et L' , respectivement, nous pouvons trouver des grammaires générant les langages $L \cup L'$, \overline{L} , $L \cap L'$, $L \cdot L'$ et L^* à partir des grammaires G et G' . Trouver ces opérations de composition pour trouver $L \cup L'$, $L \cdot L'$ et L^* fait l'objet de l'exercice 133 (p. 251). Notons que pour trouver les grammaires pour \overline{L} et $L \cap L'$, il est possible de passer par les automates.*

1. Toutefois, il est possible d'utiliser un automate pour la génération de mots en utilisant un algorithme basé sur le parcours des états guidé vers les états accepteurs.

Des grammaires vers les automates

Grammaire linéaire à droite vers automate Intuitivement, nous construisons un ϵ -AFEND selon les étapes suivantes :

1. Étape de *création des états*. L'ensemble des états de l'automate est tel que :
 - nous associons un état par symbole non terminal,
 - l'axiome est l'état initial,
 - nous ajoutons un nouvel état final F .
2. Étape de *création des transitions* associées aux règles de production.
 - Pour chaque règle $A \rightarrow xB$, nous créons une transition entre l'état A et l'état B étiquetée par x
 - Pour chaque règle $A \rightarrow x$, nous créons une transition entre l'état A et l'état F étiquetée par x .

Dans la suite, nous considérons une grammaire régulière, linéaire droite $G = (V_T, V_N, Z, P)$.

Définition 153 (Automate associé à une grammaire linéaire droite) Le « *pseudo ϵ -AFEND* » associé à G est

$$(V_N \cup \{F\}, Z, V_T, \Delta, \{F\})$$

avec $F \in V_N \cup V_T$ et :

$$\begin{aligned} \Delta = & \{(N, x, N') \mid N \rightarrow x \cdot N' \in P\} \\ & \cup \{(N, x, F) \mid N \rightarrow x \in P\} \\ & \cup \{(N, \epsilon, F) \mid N \rightarrow \epsilon \in P\} \end{aligned}$$

Proposition 13 (Correction de la synthèse d'automate depuis une grammaire) Étant donnée une grammaire G , le langage reconnu par l'automate obtenu à partir de cette grammaire est le langage généré par la grammaire :

$$\mathcal{L}_{\text{gram}}(G) = \mathcal{L}_{\text{auto}}(\text{gram2auto}(G)).$$

Remarque 49 (Grammaire linéaire à gauche vers automate) Nous pouvons obtenir un automate à partir d'une grammaire linéaire à gauche en réutilisant la transformation des grammaires linéaires à droite, comme suit :

- On transforme la grammaire linéaire gauche en grammaire linéaire droite en prenant le miroir des parties droites des règles.
- On applique la transformation utilisée pour les grammaires linéaires droites pour obtenir un automate intermédiaire.
- Dans l'automate intermédiaire :
 - on inverse les transitions;
 - l'état initial est celui correspondant à l'axiome;
 - les états finaux sont ceux avec des règles avec une partie droite constituée d'un symbole non terminal.

Nous obtenons ainsi l'automate final.

Des automates vers les grammaires

Dans la suite, nous considérons un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$.

L'intuition de la transformation est la suivante :

- L'ensemble des symboles terminaux est l'alphabet de l'automate.
- Associer un symbole non terminal par état.
- L'axiome est l'état initial.
- Les états accepteurs sont associés à des règles de production où la partie droite est ϵ .
- Pour chaque transition de q vers q' sur un symbole a , alors on a une règle de production $q \rightarrow a \cdot q'$.

Définition 154 (Grammaire régulière associée à un automate) *La grammaire associée à A est $(Q, \Sigma, q_{\text{init}}, P)$ avec :*

$$P = \{(q, a \cdot q') \mid (q, a, q') \in \delta\} \cup \{(f, \epsilon) \mid f \in F\}.$$

Proposition 14 (Correction de la traduction des automates vers les grammaires) *Étant donné un automate A , le langage généré par la grammaire obtenu à partir A est le langage reconnu par A :*

$$\mathcal{L}_{\text{auto}}(A) = \mathcal{L}_{\text{gram}}(\text{auto2gram}(A)).$$

Remarque 50 *Cette transformation s'adapte aisément pour générer des grammaires à partir d'AFEND et ϵ -AFEND.*

12.2.2 Grammaires et expressions régulières

Traduction (compositionnelle) des expressions régulières en grammaires

Définition 155 (Traduction des expressions régulières en grammaire linéaire à droite) *La traduction se fait de manière compositionnelle en suivant la syntaxe des expressions régulières.*

- Pour les expressions régulières de base :
 - pour ϵ : $(\{Z\}, \emptyset, Z, \{Z \rightarrow \epsilon\})$;
 - pour \emptyset : $(\{Z\}, \emptyset, Z, \emptyset)$;
 - pour $s \in \Sigma$: $(\{Z\}, \{s\}, Z, \{Z \rightarrow s\})$;
- Pour les expressions régulières composées : soient $G_E = (N_E, T_E, Z_E, P_E)$ et $G_{E'} = (N_{E'}, T_{E'}, Z_{E'}, P_{E'})$ des grammaires linéaires à droite pour des expressions régulières e et e' . Pour simplifier, nous supposons que $N_E \cap N_{E'} = \emptyset$.
 - Pour $e + e'$: $(N_E \cup N_{E'}, T_E \cup T_{E'}, Z, P_E \cup P_{E'} \cup \{Z \rightarrow Z_E \mid Z_{E'}\})$
 - Pour $e \cdot e'$: $(N_E \cup N_{E'}, T_E \cup T_{E'}, Z, P'_E \cup P_{E'})$,
avec $P'_E = \{A \rightarrow x \cdot Z_{E'} \mid A \rightarrow x \in P_E\} \cup \{A \rightarrow x \cdot B \in P_E\}$.
 - Pour e^* : $(N_E, T_E, Z_E, P_E \cup \{A \rightarrow x \cdot Z_E \mid A \rightarrow x \in P_E\} \cup \{Z_E \rightarrow \epsilon\})$

12.3 Exercices

12.3.1 Des grammaires vers les automates

Exercice 134 (♠♦) — Grammaire vers automate

Si cela est possible, donner des automates qui reconnaissent les langages générés par les grammaires suivantes. Si cela n'est pas possible, justifier.

1. Les grammaires données dans l'exercice 129,
2. Les grammaires données dans l'exercice 130,

12.3.2 Des automates vers les grammaires

Exercice 135 (♣) — Composition de grammaires, avec les automates

Soient L et L' des langages réguliers générés par les grammaires G et G' , respectivement. On souhaite démontrer qu'il existe des grammaires régulières qui génèrent les langages suivants : \bar{L} , $L \cap L'$. (Remarquons que pour construire $L \cap L'$ et \bar{L} , il est possible de passer par les automates.)

1. Indiquer comment utiliser les transformations entre les grammaires et automates pour obtenir une grammaire générant \bar{L} et $L \cap L'$ à partir de G et G' .

Exercice 136 (♠♦) — Des AFED vers les grammaires

Donner des grammaires générant les langages reconnus par les AFED donnés dans les figures suivantes.

1. Les AFED dans la figure 3.2 (p. 49).
2. Les AFED dans la figure 10.4 (p. 229).
3. Les AFED non minimaux dans la figure 6.2 (p. 127).

Exercice 137 (♠♦) — Des AFEND vers les grammaires

Donner les grammaires générant les langages reconnaissant par les AFEND donnés dans les figures suivantes.

1. Les AFEND dans la figure 7.2 (p. 146).
2. Les AFEND dans la figure 7.3 (p. 149).

Exercice 138 (♠♦) — Des automates vers les grammaires

Considérons les AFEND dans la figure 7.2 (p. 146) et figure 7.3 (p. 149) et les AFED correspondants obtenus dans le chapitre 7.

1. Donner une grammaire correspondante à chacun des AFED.
2. Comparer la grammaire obtenu à la question précédente aux grammaires obtenues dans l'exercice 137 (p. 261).

12.3.3 Des expressions régulières vers les automates et les grammaires

Exercice 139 (♠♦) — Des expressions régulières vers les grammaires

Pour chacun des langages suivants décrits par les expressions régulières, donner un AFEND qui reconnaît le langage et utiliser l'AFEND pour obtenir une grammaire générant ce langage.

- | | |
|-------------------|---------------------------------------------------|
| 1. $a \cdot b.$ | 5. $(a + b)^* \cdot a^* \cdot b^* \cdot a.$ |
| 2. $a^*.$ | 6. $a + (a + b)^* \cdot a \cdot b^*.$ |
| 3. $a^* \cdot b.$ | 7. $\emptyset \cdot b^* + (a \cdot \emptyset)^*.$ |
| 4. $(a + b)^*.$ | |

12.3.4 Grammaires non régulières

Exercice 140 (♣) — Langage généré par une grammaire

Indiquer les langages générés par les grammaires de la forme $(\{a, b\}, \{Z, T, U\}, Z, P)$ avec les ensembles de règles de production suivants P :

- | | |
|-----------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| 1. $\left\{ \begin{array}{l} Z \rightarrow T \mid bZb, \\ T \rightarrow aT \mid \epsilon \end{array} \right\}.$ | 2. $\{ Z \rightarrow \epsilon \mid aZb \}.$ |
|-----------------------------------------------------------------------------------------------------------------|---------------------------------------------|

Exercice 141 (♠♦) — Grammaire qui génère un langage

Pour chacun des langages suivants, donner une grammaire qui le génère et générer quelques mots. Indiquer si cette grammaire est régulière, linéaire à droite ou linéaire à gauche.

1. $\{ a^n \cdot b^{2 \times n} \mid n \in \mathbb{N} \}.$
2. $\{ a^n \cdot b^n \mid n \in \mathbb{N} \}.$
3. $\{ a^n \cdot b \cdot c^n \mid n \in \mathbb{N} \}.$
4. $\{ a^{n+1} \cdot b^n \mid n \in \mathbb{N} \}.$

Exercice 142 (♠♦) — Grammaire générant les expressions booléennes

Nous considérons des expressions booléennes simples formées par des variables, la composition par les opérateurs de disjonction, de conjonction, et de négation.

1. Donner une grammaire qui permet de générer des expressions booléennes telles que décrites ci-dessus.

Exercice 143 (♠♦) — Grammaire générant les commandes conditionnelles

Nous considérons un langage de programmation qui contient des commandes dans l'ensemble Σ_{stm} .

1. Donner une grammaire qui permet de générer des commandes conditionnelles de la forme `if ...then ...`

2. Donner une grammaire qui permet de générer des commandes conditionnelles de la forme `if ...then ...else ...`

Exercice 144 (♠♠) — Grammaire générant les expressions régulières

Nous considérons les expressions régulières, comme définies en cours.

1. Donner une grammaire qui permette de générer les expressions régulières.

Exercice 145 (♠♠) — Lien entre non-régularité d'un langage et d'une grammaire

Nous considérons le langage $\{a^n \cdot b^n \mid n \in \mathbb{N}\}$. Ce langage n'est pas régulier.

1. Essayer d'écrire une grammaire régulière qui génère ce langage et décrire le problème qui se pose.

12.3.5 Composition de grammaires régulières

Exercice 146 (♠♠) — Composition de grammaires

Soient L et L' des langages réguliers générés par les grammaires G et G' respectivement. On souhaite démontrer qu'il existe des grammaires régulières qui génèrent les langages suivants : $L \cup L'$, $L \cdot L'$, $(L)^*$.

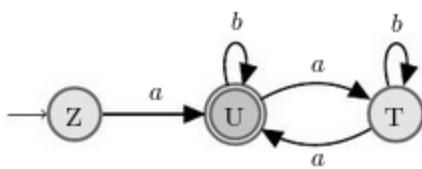
1. Donner les grammaires pour les langages demandés à partir de grammaires régulières quelconques. Expliquer la construction de ces grammaires.

12.4 Solutions des exercices

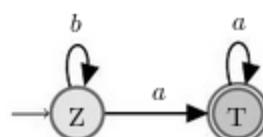
12.4.1 Des grammaires vers les automates

Solution de l'exercice 134 (page 261)

1. Ces grammaires sont régulières à droite. Nous appliquons la transformation décrite dans la définition 153 (p. 259).
 - Nous proposons l'automate représenté dans la figure 12.1a.



(a) Pour la première grammaire.



(b) Pour la seconde grammaire.

Figure 12.1 – Automates solutions de l'exercice 134, pour les grammaires données dans l'exercice 129.

- Nous proposons l'automate représenté dans la figure 12.1b.
- Cela est impossible, les grammaires ne sont pas régulières ; elles ne sont ni linéaire à droite ni linéaire à gauche. Notons que cela ne signifie pas qu'il est impossible, pour certaines de ces grammaires, de les transformer en grammaire régulière.

12.4.2 Des automates vers les grammaires

Solution de l'exercice 135 (page 261)

- À partir de G et G' , nous obtenons deux automates A et A' reconnaissant L et L' respectivement. Nous pouvons utiliser les transformations de l'automate complémentaire et de l'automate produit pour obtenir $A_{\overline{L}}$ et $A_{L \cap L'}$ qui reconnaissent \overline{L} et $L \cap L'$ respectivement ; voir définition 79 (p. 72) et définition 80. Ensuite, nous pouvons utiliser les transformations depuis les automates vers les grammaires sur $A_{\overline{L}}$ et $A_{L \cap L'}$ pour obtenir les grammaires recherchées.

Solution de l'exercice 136 (page 261)

Pour les questions suivantes, nous donnons une grammaire linéaire à droite en décrivant la grammaire par son ensemble de règles de production. Nous associons le symbole non terminal U (resp. D, T, Q) à l'état 1 (resp. 2, 3, 4) de l'automate. Le symbole non terminal U est l'axiome.

- Pour les AFED de la figure 3.2a à la figure 3.2f, nous associons les grammaires représentées dans les figure 12.2a à figure 12.2f.
- Pour l'AFED dans la figure 10.6b (resp. figure 10.6c), nous associons la grammaire représentée dans la figure 12.2g (resp. figure 12.2h).
- Pour l'AFED dans la figure 6.2a (resp. figure 6.2b), nous associons la grammaire représentée dans la figure 12.2i (resp. figure 12.2j).

Solution de l'exercice 137 (page 261)

Pour les questions suivantes, nous donnons une grammaire linéaire à droite en décrivant la grammaire par son ensemble de règles de production. Nous associons le symbole non terminal U (resp. D, T, Q) à l'état 1 (resp. 2, 3, 4) de l'automate. Le symbole non terminal U est l'axiome.

- Pour l'AFEND dans la figure 7.2a (resp. figure 7.2b, figure 7.2c), nous associons la grammaire représentée dans la figure 12.3d (resp. figure 12.3e, figure 12.3c).
- Pour l'AFEND dans la figure 7.3a (resp. figure 7.3b), nous associons la grammaire représentée dans la figure 12.3d (resp. figure 12.3e).

Solution de l'exercice 138 (page 261)

- Nous partons de la représentation tabulaire des automates obtenus et nous les représentons sous forme graphique en faisant un renommage des états. Dans les grammaires, le symbole non terminal A est l'axiome.

— Nous considérons les automates de la figure 7.2.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot T, \\ D \rightarrow a \cdot D \mid b \cdot T \mid \epsilon, \\ T \rightarrow a \cdot T \mid b \cdot T \end{array} \right\}$$

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot U, \\ D \rightarrow a \cdot T \mid b \cdot D, \\ T \rightarrow a \cdot T \mid b \cdot T \mid \epsilon \end{array} \right\}$$

(a) Pour l'automate dans la figure 3.2a.

(b) Pour l'automate dans la figure 3.2b.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D, \\ D \rightarrow a \cdot T, \\ T \rightarrow a \cdot T \mid b \cdot T \mid \epsilon \end{array} \right\}$$

$$\left\{ \begin{array}{l} Z \rightarrow a \cdot U \mid b \cdot Z, \\ U \rightarrow a \cdot D \mid b \cdot U \mid \epsilon, \\ D \rightarrow a \cdot Z \mid b \cdot D \end{array} \right\}$$

(c) Pour l'automate dans la figure 3.2c.

(d) Pour l'automate dans la figure 3.2d.

$$\left\{ \begin{array}{l} Z \rightarrow a \cdot U, \\ U \rightarrow a \cdot D \mid \epsilon, \\ D \rightarrow a \cdot Z \end{array} \right\}$$

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot T, \\ D \rightarrow a \cdot T \mid b \cdot U \mid \epsilon, \\ T \rightarrow a \cdot U \mid b \cdot D \end{array} \right\}$$

(e) Pour l'automate dans la figure 3.2e.

(f) Pour l'automate dans la figure 3.2f.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot U, \\ D \rightarrow a \cdot T \mid b \cdot D, \\ T \rightarrow a \cdot T \mid b \cdot D \mid \epsilon \end{array} \right\}$$

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot T, \\ D \rightarrow a \cdot U \mid b \cdot T, \\ T \rightarrow a \cdot D \mid b \cdot U \mid \epsilon \end{array} \right\}$$

(g) Pour l'automate dans la figure 10.6b.

(h) Pour l'automate dans la figure 10.6c.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot U, \\ D \rightarrow a \cdot D \mid b \cdot T, \\ T \rightarrow a \cdot Q \mid b \cdot T \mid \epsilon, \\ Q \rightarrow a \cdot Q \mid b \cdot T \mid \epsilon \end{array} \right\}$$

$$\left\{ \begin{array}{l} U \rightarrow a \cdot D \mid b \cdot U, \\ D \rightarrow a \cdot D \mid b \cdot Q, \\ T \rightarrow a \cdot T \mid b \cdot U, \\ Q \rightarrow a \cdot T \mid b \cdot Q \mid \epsilon \end{array} \right\}$$

(i) Pour l'automate dans la figure 6.2a.

(j) Pour l'automate dans la figure 6.2b.

Figure 12.2 – Grammaires solutions de l'exercice 136 (p. 261). Dans ces grammaires, le symbole non terminal dans la première règle de production est l'axiome.

- Nous considérons l'automate obtenu à partir de l'AFEND dans figure 7.2a. Cet automate est représenté dans la figure 12.4a et la figure 12.4b. Nous associons à cet automate la grammaire représentée par l'ensemble de règles de production dans la figure 12.4c.
- Nous considérons l'automate obtenu à partir de l'AFEND dans figure 7.2b. Cet automate est représenté figure 12.4d et la figure 12.4e. Nous associons à cet automate la grammaire représentée par l'ensemble de règles de production dans la figure 12.4f.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot U \mid a \cdot D \mid b \cdot U, \\ D \rightarrow a \cdot T \mid b \cdot T, \\ T \rightarrow \epsilon \end{array} \right\} \left\{ \begin{array}{l} U \rightarrow b \cdot D \mid b \cdot T \mid \epsilon, \\ D \rightarrow a \cdot U \mid b \cdot T \end{array} \right\} \left\{ \begin{array}{l} U \rightarrow a \cdot U \mid a \cdot D, \\ D \rightarrow b \cdot D \mid b \cdot T, \\ T \rightarrow \epsilon \end{array} \right\}$$

(b) Pour l'AFEND dans la fi-

- (a) Pour l'AFEND dans la fi- gure 7.2a.
- (c) Pour l'AFEND dans la fi- gure 7.2c.

$$\left\{ \begin{array}{l} U \rightarrow a \cdot T \mid b \cdot D \mid \epsilon \\ D \rightarrow a \cdot U \mid a \cdot T \mid b \cdot T, \\ T \rightarrow a \cdot T \mid b \cdot T, \end{array} \right\}$$

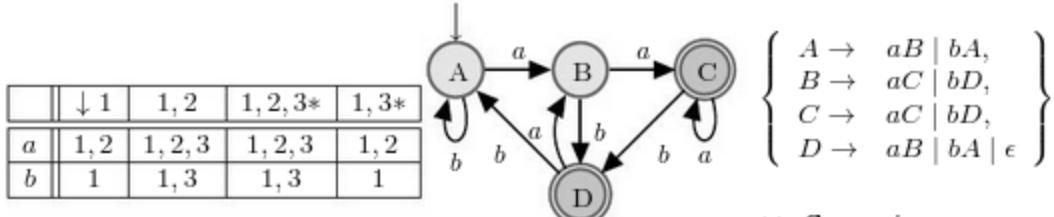
$$\left\{ \begin{array}{l} U \rightarrow b \cdot D \mid b \cdot T \mid \epsilon, \\ D \rightarrow a \cdot U \mid b \cdot T \end{array} \right\}$$

- (d) Pour l'AFEND dans la figure 7.3a.

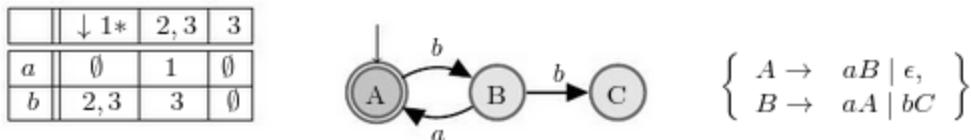
(e) Pour l'AFEND dans la figure 7.3b. Il n'y a pas de règle associée à l'état 3.

Figure 12.3 – Grammaires solutions de l'exercice 137 (p. 261). Dans ces grammaires, le symbole non terminal U est l'axiome.

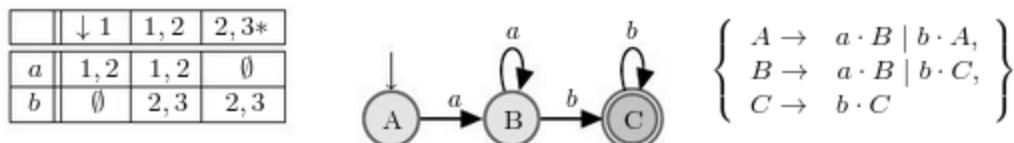
- Nous considérons l'automate obtenu à partir de l'AFEND dans figure 7.2c. Cet automate est représenté dans la figure 12.4g et figure 12.4h. Nous associons à cet automate la grammaire représentée par l'ensemble de règles de production dans la figure 12.4i.
 - Nous considérons les automates de la figure 7.3.
 - Nous considérons l'automate obtenu à partir de l'AFEND dans figure 7.3a. Cet automate est représenté dans la figure 12.4j et figure 12.4k. Nous associons à cet automate la grammaire représentée par l'ensemble de règles de production dans la figure 12.4l.
 - Nous considérons l'automate obtenu à partir de l'AFEND dans figure 7.3b. Cet automate est représenté dans la figure 12.4m et figure 12.4n. Nous associons à cet automate la grammaire représentée par l'ensemble de règles de production dans la figure 12.4o.
2. Tout d'abord, nous notons que nous savons que ces grammaires sont équivalentes (c'est-à-dire elles génèrent le même langage) car elles ont été obtenues à partir d'automates équivalents.
- Sans pouvoir exprimer de règle générale, nous pouvons observer que les grammaires générées à partir des versions non déterministes ont moins de symboles non terminaux mais plus de parties droites possibles pour un symbole non terminal donné. Ceci peut se comprendre par le fait que la transformation depuis les automates vers les grammaires associe un symbole non terminal à chaque état de l'automate. De plus, les automates non déterministes sont généralement plus concis que leur déterminisé.



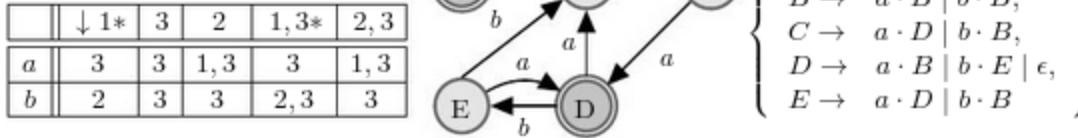
- (a) AFED obtenu à partir de l'AFEND (b) Renommage des états de l'AFED dans la figure 7.2a. (c) Grammaire correspondant à l'AFED dans la figure 12.4b.



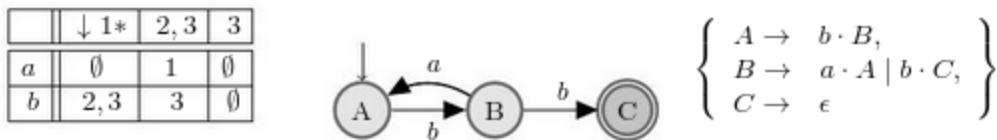
- (d) AFED obtenu à partir de l'AFEND dans la figure 7.2b. (e) Renommage des états de l'AFED dans la figure 12.4d. (f) Grammaire correspondant à l'AFED dans la figure 12.4e.



- (g) AFED obtenu à partir de l'AFEND dans la figure 7.2c. (h) Renommage des états de l'AFED dans la figure 12.4g. (i) Grammaire correspondant à l'AFED dans la figure 12.4h.



- (j) AFED obtenu à partir de l'AFEND dans la figure 7.3a. (k) Renommage des états de l'AFED dans la figure 12.4j. (l) Grammaire correspondant à l'AFED dans la figure 12.4k.



- (m) AFED obtenu à partir de l'AFEND dans la figure 7.3b. (n) Renommage des états de l'AFED dans la figure 12.4j. (o) Grammaire correspondant à l'AFED dans la figure 12.4k.

Figure 12.4 – Automates et grammaires solutions de l'exercice 138 (p. 261). Dans les grammaires, le symbole non terminal A est l'axiome.

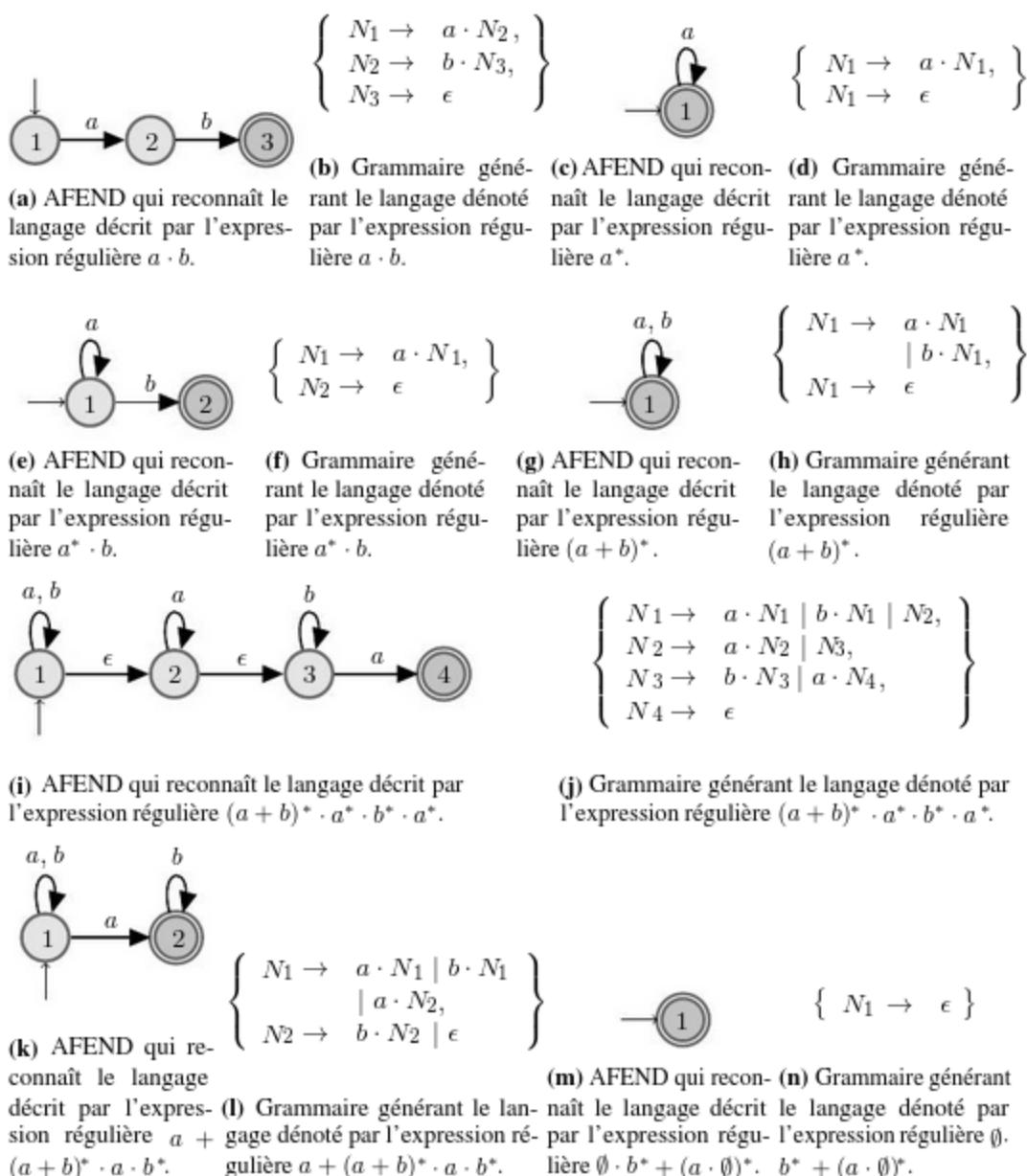


Figure 12.5 – AFEND et grammaires solutions de l'exercice 139 (p. 262). Dans les grammaires, l'axiome est N_1 .

12.4.3 Des expressions régulières vers les automates et les grammaires

Solution de l'exercice 139 (page 262)

1. L'AFEND dans la figure 12.5a reconnaît le langage décrit par l'expression régulière $a \cdot b$. Une grammaire équivalente est décrite par l'ensemble des règles de production dans la figure 12.5b.
2. L'AFEND dans la figure 12.5c reconnaît le langage décrit par l'expression régulière a^* . Une grammaire équivalente est décrite par l'ensemble des règles de production dans la figure 12.5d.
3. L'AFEND dans la figure 12.5e reconnaît le langage décrit par l'expression régulière $a^* \cdot b$. Une grammaire équivalente est donnée par l'ensemble des règles de production dans la figure 12.5f.
4. L'AFEND dans la figure 12.5g reconnaît le langage décrit par l'expression régulière $(a+b)^*$. Une grammaire équivalente est donnée par l'ensemble des règles de production dans la figure 12.5h.
5. L'AFEND dans la figure 12.5i reconnaît le langage décrit par l'expression régulière $(a+b)^* \cdot a^* \cdot b^* \cdot a$. Une grammaire équivalente est donnée par l'ensemble des règles de production dans la figure 12.5j.
6. L'AFEND dans la figure 12.5k reconnaît le langage décrit par l'expression régulière $a + (a+b)^* \cdot a \cdot b^*$ qui est équivalente à $(a+b)^* \cdot a \cdot b^*$. Une grammaire équivalente est donnée par l'ensemble des règles de production dans la figure 12.5l.
7. L'AFEND dans la figure 12.5m reconnaît le langage décrit par l'expression régulière $\emptyset \cdot b^* + (a \cdot \emptyset)^*$ qui est équivalente à \emptyset^* et donc ϵ . Une grammaire équivalente est donnée par l'ensemble des règles de production dans la figure 12.5n.

12.4.4 Grammaires non régulières

Solution de l'exercice 140 (page 262)

1. Le langage généré par cette grammaire est : $\{b^i \cdot a^j \cdot b^i \mid i, j \in \mathbb{N}\}$. C'est l'ensemble des mots commençant par des b et finissant par des b et avec des a entre les deux, de plus le nombre de b au début et en fin de mot doivent être les mêmes.
2. Le langage généré par cette grammaire est : $\{a^n b^n \mid n \in \mathbb{N}\}$. C'est l'ensemble des mots commençant par des a et finissant par des b , avec autant d'occurrences du symbole a que du symbole b et aucun symbole entre les a et les b .

Solution de l'exercice 141 (page 262)

Nous donnons directement l'ensemble des règles de production de la grammaire ; ceci définit les symboles non terminaux et les terminaux de la grammaire, ainsi que l'axiome qui est le symbole non terminal utilisé dans la première règle de dérivation. Ainsi, dans les ensembles de règles de production suivants les symboles Z , A , B sont des symboles non terminaux, les symboles a , b et c sont des terminaux, le symbole Z est l'axiome. Pour certains langages, nous proposons plusieurs solutions.

1. Nous proposons les trois grammaires équivalentes suivantes :

$$\begin{aligned} & - \{Z \rightarrow aZbb \mid \epsilon\}; \\ & - \left\{ \begin{array}{l} Z \rightarrow AZB \mid \epsilon, \\ A \rightarrow a, \\ B \rightarrow bb \end{array} \right\}; \\ & - \left\{ \begin{array}{l} Z \rightarrow ASBB \mid \epsilon, \\ A \rightarrow a, \\ B \rightarrow b \end{array} \right\}. \end{aligned}$$

2. Nous proposons les trois grammaires équivalentes suivantes :

$$\begin{aligned} & - \{Z \rightarrow aZb \mid \epsilon\}; \\ & - \left\{ \begin{array}{l} Z \rightarrow AZB \mid \epsilon, \\ A \rightarrow a, \\ B \rightarrow b \end{array} \right\}; \\ & - \left\{ \begin{array}{l} Z \rightarrow AB \mid AZB, \\ A \rightarrow a \mid \epsilon, \\ B \rightarrow b \mid \epsilon \end{array} \right\}. \end{aligned}$$

3. Nous proposons la grammaire suivante : $\{Z \rightarrow b \mid abc \mid aZc\}$.

4. Nous proposons la grammaire suivante : $\left\{ \begin{array}{l} Z \rightarrow aS, \\ S \rightarrow \epsilon \mid aSb \end{array} \right\}$.

Solution de l'exercice 142 (page 262)

1. Nous proposons la grammaire suivante : $\{E \rightarrow Var \mid E \vee E \mid E \wedge E \mid \neg E\}$.

Solution de l'exercice 143 (page 262)

1. Nous proposons la grammaire suivante : $\{Z \rightarrow \text{if } B \text{ then } Z \mid \Sigma_{\text{stm}}\}$.
 2. Nous proposons la grammaire suivante : $\{Z \rightarrow \text{if } B \text{ then } Z \text{ else } Z \mid \Sigma_{\text{stm}}\}$.

Solution de l'exercice 144 (page 263)

1. Nous proposons la grammaire suivante : $\{E \rightarrow \Sigma \mid \emptyset \mid \epsilon \mid E + E \mid E \cdot E \mid E^*\}$.

Solution de l'exercice 145 (page 263)

1. En écrivant la grammaire, nous nous rendons compte qu'il faut écrire une règle et un symbole non terminal par mot du langage, car pour chaque mot il existe un langage unique de mots qui peuvent continuer le mot préfixe pour former un mot du langage :

$$\left\{ \begin{array}{l} Z \rightarrow aB_1 \mid aaB_2 \mid aaaB_3 \mid \dots, \\ B_1 \rightarrow b, \\ B_2 \rightarrow bb, \\ B_3 \rightarrow bbb \\ \dots \end{array} \right\}$$

12.4.5 Composition de grammaires régulières

Solution de l'exercice 146 (page 263)

- Soient (T, N, Z, P) et (T', N', Z', P') les grammaires pour L et L' . Supposons que ces grammaires soient linéaires à droite. Nous supposons que les ensembles N et N' sont disjoints, ce qui est toujours possible après un renommage des symboles non terminaux. Soit $Z'' \notin N \cup N'$, un nouveau symbole non terminal.

— La grammaire reconnaissant $L \cup L'$ est :

$$(T \cup T', N \cup N' \cup \{Z''\}, Z'', P \cup P' \cup \{Z'' \rightarrow Z \mid Z'\}).$$

Nous avons considéré l'union des symboles non terminaux, terminaux et règles de production. Nous avons également ajouté le symbole non terminal Z'' qui devient l'axiome de la nouvelle grammaire et la règle de production $Z'' \rightarrow Z \mid Z'$ introduisant le choix entre la génération d'un mot généré par l'une des grammaires.

— La grammaire reconnaissant $L \cdot L'$ est :

$$(T \cup T', N \cup N', Z'', P'' \cup P'),$$

avec $P'' = \{N_t \rightarrow x \cdot N'_t \in P \mid x \neq \epsilon\} \cup \{N_t \rightarrow Z' \mid N_t \rightarrow x \in P\}$.

Nous avons considéré l'union des symboles non terminaux et terminaux. Nous avons également ajouté le symbole non terminal Z'' qui devient l'axiome de la nouvelle grammaire. Pour les règles de production, nous gardons les règles de production de G' et modifions les règles de production de G . Pour chaque règle de production dans P de la forme $N_t \rightarrow x \cdot N'_t$, nous examinons le mot x de terminaux préfixant le symbole non terminal N'_t :

- si ce mot x est différent de ϵ , cette règle est gardée ;
- sinon (ce mot x est ϵ), nous mettons en partie droite l'axiome de G' , la règle devenant $N_t \rightarrow x \cdot N'_t$.

— La grammaire reconnaissant L^* est :

$$(T, N, Z, P'),$$

avec $P' = P \cup \{Z \rightarrow \epsilon\} \cup \{N_t \rightarrow x \cdot Z \mid N_t \rightarrow x \in P\}$.

Nous avons considéré les symboles non terminaux et terminaux de G . Pour les règles de production, nous gardons les règles de production de G et ajoutons la règle $Z \rightarrow \epsilon$. Pour chaque règle de production dans P de la forme $N_t \rightarrow x$, nous ajoutons la règle $N_t \rightarrow x \cdot Z$.

Nous observons que les grammaires proposées sont linéaires à droite.

Propriété de l’itération

13.1 Résumé intuitif du chapitre

Lorsqu’un automate contient un cycle, le chemin qui l’étiquette peut être itéré un nombre quelconque de fois. Cela se traduit par le fait que qu’un mot w suffisamment grand, typiquement dont la longueur est supérieur au nombre d’états, peut se décomposer en 3 parties x, y et z telles que $w = x \cdot y \cdot z$ et $\forall n \in \mathbb{N}, x \cdot y^n \cdot z$ est dans le langage ; y correspondant au mot qui étiquette le cycle. Les conditions sur x, y et z sont explicitées. Cette propriété est donnée par le **lemme de l’itération**. Tout langage régulier satisfait le lemme de l’itération. Cependant, certains langages non réguliers satisfont également le lemme de l’itération. Nous utiliserons néanmoins le lemme de l’itération dans le chapitre suivant pour démontrer que des langages ne sont pas réguliers.

De plus, on peut s’attacher à trouver la **constante d’itération minimale** d’un langage qui est le plus petit entier qui détermine la longueur des mots de ce langage à partir de laquelle une partie de ces mots peut être itérée comme décrit par le lemme de l’itération.

13.2 Les notions essentielles

13.2.1 Lemme de l’itération

Dans la suite, nous considérons un langage régulier L .

Lemme 6 (Lemme de l’itération) *Il existe $N \in \mathbb{N}$ tel que pour tout mot $w \in L$ avec $|w| \geq N$, on peut trouver $x, y, z \in \Sigma^*$ tels que $w = x \cdot y \cdot z$ et :*

1. $y \neq \epsilon$,
2. $|x \cdot y| \leq N$,
3. pour tout $k \in \mathbb{N}$, $x \cdot y^k \cdot z \in L$.

Le lemme de l’itération est illustré sur la figure 13.1.

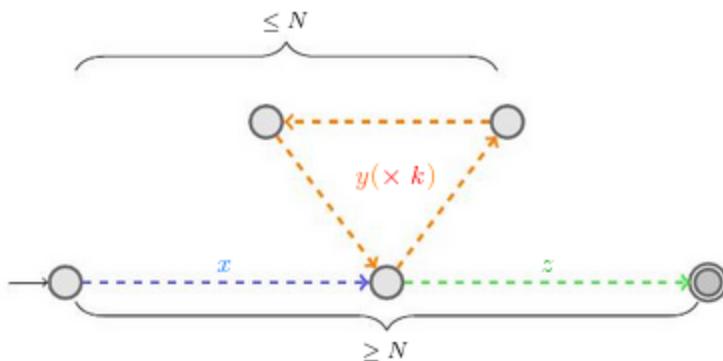


Figure 13.1 – Illustration du lemme de l’itération. Un mot de longueur supérieure ou égale à une certaine constante N et accepté par l’automate peut se décomposer en trois facteurs x , y et z tels que $y \neq \epsilon$, $|x \cdot y| \leq N$ et pour tout $k \in \mathbb{N}$, $x \cdot y^k \cdot z \in L$.

Nous utiliserons le lemme de l’itération pour démontrer qu’un langage est non régulier. Pour cela, nous ferons des preuves par l’absurde. Plus précisément, le langage considéré sera supposé régulier. Ainsi, ce langage aura une constante d’itération N (voir définition plus précise plus loin) et nous pourrons utiliser le lemme de l’itération. Nous prendrons un mot dans le langage qui est de longueur supérieure ou égale à N . Nous pourrons déduire des propriétés sur ce mot accepté en utilisant les propriétés données par le lemme. Nous pourrons également déduire d’autres mots acceptés dans le langage, ceci afin de parvenir à une contradiction.

13.2.2 Constante d’itération

Dans cette section, nous considérons un langage régulier L (quelconque).

Définition 156 (Constante d’itération) *Selon le lemme de l’itération, il existe (au moins) un $N \in \mathbb{N}$ tel que chaque mot w de L est tel que $|w| \geq N$ possède un facteur non vide (y) qui peut être itéré.*

Un tel entier N est appelé une constante d’itération de L et le facteur est dit facteur itérant.

Les deux propriétés suivantes découlent de la définition de la constante d’itération.

Propriété 16 (À propos de la constante d’itération)

- Si N est une constante d’itération de L , alors tout $N' \geq N$ est une constante d’itération de L .
- Si N n’est pas une constante d’itération de L , alors tout $N' \leq N$ n’est pas une constante d’itération de L .

Définition 157 (Constante d’itération minimale) *La constante d’itération minimale de L est sa plus petite constante d’itération.*

13.3 Exercices

13.3.1 Lemme de l'itération

Exercice 147 (♠♦) — Vérifier l'application du lemme de l'itération

Considérer quelques automates étudiés dans le chapitre 3, par exemple des automates dans la figure 3.2.

- Monter que le lemme de l'itération s'applique sur les langages reconnus par ces automates. C'est-à-dire donner des valeurs instances des variables impliquées dans le lemme (la constante N , un mot w , la décomposition de w en trois facteurs x , y et z).

Exercice 148 (♠♦♦) — Démontrer le lemme de l'itération

- Démontrer le lemme de l'itération.

13.3.2 Constante d'itération

Exercice 149 (♠♦) — Trouver la constante d'itération minimale

Considérons l'alphabet $\Sigma = \{0, 1\}$. Donner la constante d'itération minimale des langages réguliers sur Σ dénotés par les expressions régulières suivantes :

- | | | |
|---------------|-----------------|------------------|
| 1. 0001^* . | 4. 1011 . | 7. $(0 + 1)^*$. |
| 2. 0^* . | 5. $(01)^*$. | 8. 10^*1 . |
| 3. 0^*1^* . | 6. ϵ . | |

Exercice 150 (♠♦♦) — Trouver la constante d'itération minimale

Considérons l'alphabet $\Sigma = \{0, 1\}$. Donner la constante d'itération minimale des langages réguliers sur Σ dénotés par les expressions régulières suivantes :

- | | | |
|---------------------|-----------------------------|-------------------------|
| 1. $10(11^*0)^*0$. | 3. $0^*1^+0^+1^*$. | 5. 0^*101^* . |
| 2. $011 + 0^*1^*$. | 4. $0^*1^+0^+1^* + 10^*1$. | 6. $0^*101^* + 101^*$. |

13.4 Indications pour résoudre les exercices

Indications pour l'exercice 148 (p. 275)

- Utiliser le théorème de Kleene (théorème 12 (p. 220)), le principe des tiroirs (théorème 1 (p. 24)) et l'existence d'un automate qui reconnaît ce langage.

13.5 Solutions des exercices

13.5.1 Lemme de l’itération

Solution de l’exercice 147 (page 275)

1. Nous traitons l’exercice sur deux exemples d’automates, le principe étant toujours le même.
 - Considérons l’AFED représenté dans la figure 3.2a. Considérons un entier $N \geq 2$. Un mot w de longueur supérieure ou égale à N accepté par l’automate à son exécution qui termine dans l’état 2. Ce mot s’écrit sous la forme a^n avec $n \geq 2$. Ce mot peut s’écrire sous la forme $w = x \cdot y \cdot z$, avec $x = a, y = a, z = a^{n-2}$. Nous avons $|w \cdot y| = 2, y \neq \epsilon$ et $x \cdot y^k \cdot z = a \cdot a^k \cdot a^{n-2}$ qui est accepté par l’automate et appartient donc au langage.
 - Considérons l’AFED représenté dans la figure 3.2b. Considérons un entier $N \geq 3$. Un mot w de longueur supérieure ou égale à N accepté par l’automate à son exécution qui termine dans l’état 3. L’exécution de ce mot aura pour premier état répété l’état 1, 2 ou 3.
 - Si le premier état répété est l’état 1, nous pouvons écrire ce mot sous la forme $w = x \cdot y \cdot z$, avec $x = \epsilon, y = b$ et z est le reste de w .
 - Si le premier état répété est l’état 2 (et l’état 1 n’est pas répété), nous pouvons écrire ce mot sous la forme $w = x \cdot y \cdot z$, avec $x = a, y = b$ et z est le reste de w .
 - Si le premier état répété est l’état 3 (et les états 1 et 2 ne sont pas répétés), nous pouvons écrire ce mot sous la forme $w = x \cdot y \cdot z$, avec $x = a \cdot a, y$ le troisième symbole de w et z est le reste de w .

Solution de l’exercice 148 (page 275)

1. Soit L un langage régulier. D’après le théorème de Kleene (théorème 12 (p. 220)), il existe un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ tel que $\mathcal{L}_{\text{auto}}(A) = L$. Soit $N = |Q|$ (le nombre d’états de A). Considérons un mot w dans le langage de longueur plus grande que $N : w = a_1 \cdots a_m \in L$ tel que $|w| = m \geq N$. Soit $p_i = \delta^*(q_{\text{init}}, w_{[...i]})$ l’état atteint en lisant le préfixe $w_{[...i]} = a_1 \cdots a_i$ de longueur i de w , pour $i \leq m$. Notons que tous les états p_i sont bien définis pour $i \leq m$ car le mot w est accepté par l’automate. Considérons l’application $[0, m] \rightarrow Q$ définie par $i \mapsto \delta^*(q_{\text{init}}, w_{[...i]})$, nous avons $|[0, m]| \geq N + 1 > N = |Q|$. D’après le principe des tiroirs (théorème 1 (p. 24)), il existe i et j avec $0 \leq i < j \leq N$ tels que $p_i = p_j$.

Posons $x = a_1 \cdots a_i$, $y = a_{i+1} \cdots a_j$ et $z = a_{j+1} \cdots a_m$. Alors, on a :

1. $w = x \cdot y \cdot z$, par définition de x, y et z ;
2. $y \neq \epsilon$, car $i < j$;
3. $|x \cdot y| \leq N$, car $|x \cdot y| = j - i \leq N$;
4. $x \cdot y^k \cdot z \in L$ pour tout $k \geq 0$, car $\delta^*(p_i, y) = p_j = p_i$ et donc $\delta^*(p_i, y^k) = p_i$.

13.5.2 Constante d'itération

Solution de l'exercice 149 (page 275)

- La constante d'itération minimale du langage dénoté par 0001^* est 4.

Observons d'abord que le mot 000 est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 3. Considérons donc les mots de longueur supérieure ou égale à 4. Soit w un mot dans le langage et de longueur supérieure ou égale à 4. Le mot w s'écrit 0001^{k_1} , avec $k_1 \geq 1$ et on peut lui associer la décomposition $x = 000$, $y = 1$, $z = 1^{k_1-1}$ qui satisfait les conditions du lemme de l'itération.

- La constante d'itération minimale du langage dénoté par 0^* est 1.

Observons d'abord que le mot ϵ est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 0. Considérons donc les mots de longueur supérieure ou égale à 1. Un mot $w \in 0^*$ tel que $|w| \geq 1$ s'écrit $w = 0 \cdot 0^k$, avec $k \geq 0$. Un tel mot peut être itéré par la décomposition $w = xyz$ avec $x = \epsilon$, $y = 0$, $z = 0^k$.

- La constante d'itération minimale du langage dénoté par 0^*1^* est 1.

Observons d'abord que le mot ϵ est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 0. Considérons donc les mots de longueur supérieure ou égale à 1. Soit w un mot dans le langage et de longueur supérieure ou égale à 1. Le mot w peut s'écrire $0^{k_1}1^{k_2}$ avec $k_1 + k_2 \geq 1$. Nous pouvons distinguer deux cas. Le premier cas est celui où w contient au moins une occurrence du symbole 0. Dans ce cas, le mot w peut être décomposé en $x = \epsilon$, $y = 0$, $z = 0^{k_1-1}1^{k_2}$ et nous avons bien $|xy| \leq 1$. Le second cas est celui où w ne contient pas d'occurrences du symbole 0. Ainsi, comme $|w| \geq 1$, w contient au moins une occurrence du symbole 1. Dans ce cas, le mot w peut être décomposé en $x = \epsilon$, $y = 1$, $z = 1^{k_2-1}$ et vous avez bien $|xy| \leq 1$.

- La constante d'itération minimale du langage dénoté par 1011 est 5.

Observons que le seul mot du langage est le mot 1011 qui est de longueur 4. Ce mot ne peut pas être itéré. En particulier, il n'y a pas de mot de longueur supérieure ou égale à 5 dans le langage. Ainsi, de manière triviale tous les mots de longueur supérieure ou égale à 5 dans le langage peuvent être itérés.

- La constante d'itération minimale du langage dénoté par $(01)^*$ est 2.

Observons d'abord que le mot ϵ est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 0. Considérons donc les mots de longueur supérieure ou égale à 1. Notons qu'il n'y a pas de mot de longueur 1 dans le langage, donc le lemme de l'itération s'applique de manière triviale pour les mots de longueur 1. Soit w un mot dans le langage et de longueur supérieure ou égale à 2.

Le mot w peut s'écrire $(01)^m$, avec $m \geq 1$, c'est-à-dire sous la forme $01(01)^{m'}$, avec $m' \geq 0$. En particulier, il n'y a pas de facteur itérant de longueur 1 possible. Le seul choix possible du facteur itérant est ici 01 et ce choix est toujours possible. Nous pouvons donc écrire (décomposer) w en $w = xyz$, avec $x = \epsilon$, $y = 01$, $z = 01(01)^{m'}$. Cette décomposition respecte les conditions du lemme de l'itération.

- La constante d'itération minimale du langage dénoté par ϵ est 1.

Observons que le seul mot du langage est le mot ϵ qui est de longueur 0. Ce mot ne

peut pas être itéré. En particulier, il n'y a pas de mot de longueur supérieure ou égale à 1 dans le langage. Ainsi, de manière triviale tous les mots de longueur supérieure ou égale à 1 dans le langage peuvent être itérés.

7. La constante d’itération minimale du langage dénoté par $(0 + 1)^*$ est 1.

Observons d’abord que le mot ϵ est dans le langage, mais ne peut pas être itéré. La constante d’itération minimale ne peut donc pas être 0. Considérons donc les mots de longueur supérieure ou égale à 1. Soit w un mot dans le langage et de longueur supérieure ou égale à 1. Le mot w peut s’écrire $w = 0 \cdot w'$ ou $w = 1 \cdot w'$, avec $w' \in \mathcal{L}_{\text{ER}}((0 + 1)^*)$. Dans le premier (respectivement second) cas, nous prenons la décomposition $x = \epsilon$, $y = 0$ (respectivement $y = 1$) et $z = w'$. Dans les deux cas, la décomposition choisie satisfait les conditions du lemme de l’itération.

8. La constante d’itération minimale du langage dénoté par 10^*1 est 3.

Observons d’abord que le mot 11 est dans le langage, mais ne peut pas être itéré. La constante d’itération minimale ne peut donc pas être 2. Considérons donc les mots de longueur supérieure ou égale à 3. Soit w un mot dans le langage et de longueur supérieure ou égale à 3. Le mot w peut s’écrire $10^{k_1}1$, avec $k_1 \geq 1$. Le mot w peut se décomposer en $x = 1$, $y = 0$, et $z = 0^{k_1-1}1$. Cette décomposition respecte les conditions du lemme de l’itération.

Solution de l’exercice 150 (page 275)

1. La constante d’itération minimale du langage dénoté par $10(11^*0)^*0$ est 4.

Observons que le mot 100 est de longueur 3 et dans le langage mais ne peut pas être itéré. Notons qu'il n'y a pas de mot de longueur 4 dans le langage, donc le lemme de l’itération s’applique de manière triviale pour les mots de longueur 4. Soit w un mot dans le langage et de longueur supérieure ou égale à 5. Le mot w peut s’écrire $10(1^{k_1}0)^{k_2}0$, avec $k_1, k_2 \geq 0$, $x = 10$, $y = 1$. À un tel mot w , nous pouvons associer la décomposition $x = \epsilon$, $y = 10$, $z = (1^{k_1}0)^{k_2}0$. Cette décomposition respecte les conditions du lemme de l’itération.

2. La constante d’itération minimale du langage dénoté par $011 + 0^*1^*$ est 1.

Observons d’abord que le mot ϵ est dans le langage, mais ne peut pas être itéré. La constante d’itération minimale ne peut donc pas être 0. Considérons donc les mots de longueur supérieure ou égale à 1. Soit w un mot dans le langage et de longueur supérieure ou égale à 1. D’après la sémantique des expressions régulières, soit le mot appartient à 011 soit le mot appartient à 0^*1^* . Si w appartient à 011, alors w est le mot 011 et il peut être décomposé en $x = \epsilon$, $y = 0$, $z = 11$. Nous avons bien $|xy| \leq 1$. Cette décomposition respecte les conditions du lemme de l’itération. Sinon, w appartient à 0^*1^* et le raisonnement utilisé dans la réponse précédente s’applique.

3. La constante d’itération minimale du langage dénoté par $0^*1^+0^+1^*$ est 3.

Observons d’abord que le mot 10 est dans le langage, mais ne peut pas être itéré. La constante d’itération minimale ne peut donc pas être 2. Considérons donc les mots de longueur supérieure ou égale à 3. Soit w un mot dans le langage et de longueur supérieure ou égale à 3. Nous distinguons deux cas. Si w commence par 0, alors w peut s’écrire $0^{k_1}1^{k_2}0^{k_3}1^{k_4}$, avec $k_1, k_2, k_3 > 0$ et $k_4 \geq 0$. Le mot w peut se décomposer en $x = \epsilon$, $y = 0$, $z = 0^{k_1-1}1^{k_2}0^{k_3}1^{k_4}$. Sinon, w ne commence pas par 0 et peut s’écrire $1^{k_2}0^{k_3}1^{k_4}$, avec $k_2, k_3 > 0$ et $k_4 \geq 0$. Nous distinguons deux sous-cas. Le premier

sous-cas est celui où w possède le préfixe 11. Le mot w peut se décomposer en $x = \epsilon$, $y = 1$, et z est le suffixe restant dans le mot. Le second sous-cas est celui où w ne possède pas le préfixe 11. Le mot w peut se décomposer en $x = 10$, $y = 1$ et z est le suffixe restant. Toutes les décompositions utilisées respectent les conditions du lemme de l'itération.

4. La constante d'itération minimale du langage dénoté par $0^* 1^+ 0^+ 1^* + 10^* 1$ est 3. Observons d'abord que le mot 11 est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 2. Considérons donc les mots de longueur supérieure ou égale à 3. Soit w un mot dans le langage et de longueur supérieure ou égale à 3. D'après la sémantique des expressions régulières, soit le mot appartient à $0^* 1^+ 0^+ 1^*$ soit le mot appartient à $10^* 1$.

Si w appartient à $0^* 1^+ 0^+ 1^*$, alors nous distinguons deux cas.

- Si w commence par 0, alors nous prenons la décomposition $x = \epsilon$, $y = 0$ et z est le suffixe restant de w .
- Si w ne commence pas par 0, alors nous distinguons deux sous-cas.
 - Si w contient 11 comme préfixe, alors nous prenons la décomposition $x = \epsilon$, $y = 1$ et z est le suffixe restant de w .
 - Sinon, nous avons deux nouveaux cas : soit w possède le facteur 00 et nous prenons la décomposition $x = 1$, $y = 0$ et z est le suffixe restant; sinon (w possède le préfixe 101 et nous prenons la décomposition $x = 10$, $y = 1$, z est le suffixe restant de w .

Toutes les décompositions utilisées respectent les conditions du lemme de l'itération. Si w n'appartient pas à $0^* 1^+ 0^+ 1^*$, alors il appartient à $10^* 1$. Tous les mots de longueur supérieure ou égale à 3 contiennent au moins une occurrence du facteur 0 et ces mots peuvent être décomposés comme suit : $x = 1$, $y = 0$ et z est le suffixe à partir du troisième symbole. Cette décomposition satisfait également les conditions du lemme de l'itération.

5. La constante d'itération minimale du langage dénoté par $0^* 101^*$ est 3. Observons d'abord que le mot 10 est dans le langage, mais ne peut pas être itéré. La constante d'itération minimale ne peut donc pas être 2. Considérons donc les mots de longueur supérieure ou égale à 3. Soit w un mot dans le langage et de longueur supérieure ou égale à 3. Nous distinguons deux cas. Si w possède le préfixe 0, nous prenons la décomposition $x = \epsilon$, $y = 0$ et z est le suffixe restant de w . Si w ne possède pas le préfixe 0 (alors w possède le préfixe 101), nous prenons la décomposition $x = 10$, $y = 1$ et z est le suffixe restant de w . Toutes les décompositions utilisées respectent les conditions du lemme de l'itération.

6. La constante d'itération minimale du langage dénoté par $0^* 101^* + 101^*$ est 3. Le raisonnement est le même qu'à la question précédente. On distingue simplement le cas supplémentaire au début du raisonnement où le mot appartient à 101^* . Tous les mots de longueur supérieure ou égale à 3 appartenant à ce langage s'écrivent sous la forme 101^k avec $k \geq 1$ et peuvent être décomposé en $x = 10$, $y = 1$, $z = 1^{k-1}$.

Démontrer la non-régularité

14.1 Résumé intuitif du chapitre

Nous utilisons le lemme de l’itération pour démontrer la **non-régularité d’un langage**. Plus précisément, si un langage est régulier, alors il satisfait le lemme de l’itération. Pour montrer qu’un langage n’est pas régulier, nous utilisons une technique de preuve par l’absurde : on suppose que le langage est régulier, puis on montre une contradiction en utilisant (les propriétés fournies par) le lemme de l’itération.

Avec la connaissance de la non-régularité de certains langages, comme par exemple $\{a^n \cdot b^m \mid n \in \mathbb{N}\}$, on utilise les**propriétés de fermeture** des langages réguliers (intersection, union, complément, image miroir, morphisme et morphisme inverse) afin de démontrer que d’autres langages sont non réguliers.

Enfin, nous donnons une **condition suffisante** pour démontrer de manière plus directe la non-régularité d’un langage.

14.2 Les notions essentielles

14.2.1 Utilisation du lemme de l’itération

Le lemme de l’itération (lemme 6 (p. 273)) est une propriété des langages réguliers. Autrement dit, pour un langage, satisfaire le lemme de l’itération est une *condition nécessaire* à sa régularité. Si nous notons $Régulier(L)$ la propriété indiquant qu’un langage est régulier et $Itération(L)$ la propriété indiquant qu’un langage satisfait le lemme de l’itération, le fait que le lemme de l’itération soit une condition nécessaire à la régularité s’écrit :

$$Régulier(L) \implies Itération(L)$$

Afin de démontrer qu’un langage L est non régulier en utilisant le lemme de l’itération, nous utilisons la contraposée de la proposition précédente, à savoir :

$$\neg Itération(L) \implies \neg Régulier(L)$$

puis le fait que le langage L ne satisfait pas le lemme de l’itération (c’est-à-dire $\neg\text{Itération}(L)$). Pour démontrer ce dernier point, nous utilisons la technique de preuve par l’absurde : nous supposons que le langage satisfait le lemme de l’itération et trouvons une contradiction.

Ainsi, chaque preuve de non-régularité utilisant le lemme de l’itération suit les étapes suivantes :

- Supposer que le langage est régulier ; il satisfait donc le lemme de l’itération.
- Choisir un mot w dans le langage et de longueur plus grande qu’une de ses constantes d’itération N (non connues et fournies par le lemme).
- Déduire du lemme l’existence d’une décomposition du mot en trois facteurs (x, y et z) tels que $w = x \cdot y \cdot z$ qui satisfait les propriétés données par le lemme.
- Exploiter les propriétés ($|x \cdot y| \leq N$, $y \neq \epsilon$ et $\forall k \in \mathbb{N} . x \cdot y^k \cdot z \in L$) sur la décomposition pour déduire des informations sur le mot choisi (et sa structure).
- Choisir une valeur pour le facteur itérant k qui permet d’arriver à une contradiction ; la contradiction étant de trouver un mot $x \cdot y^{k_c} \cdot z$ (avec une valeur de k égale à k_c) qui n’appartient pas au langage.

14.2.2 Utilisation des propriétés de fermeture

Dans cette section, nous utilisons les propriétés de fermeture des langages à états (c'est-à-dire réguliers) rappelées dans le tableau 8.1 (p. 173) ainsi que les langages suivants :

- $L_?$ dénote un langage dont on cherche à montrer la non-régularité,
- L_{reg} dénote un langage que l’on sait régulier,
- L_{nonreg} dénote un langage que l’on sait non régulier.

Proposition 15 (Utilisation des fermetures par union et intersection) *Si $L_? \cup L_{\text{reg}} = L_{\text{nonreg}}$ ou $L_? \cap L_{\text{reg}} = L_{\text{nonreg}}$, alors $L_?$ est non régulier. (En effet, si $L_?$ était régulier, comme les langages réguliers sont fermés par union et intersection, alors L_{nonreg} serait régulier.)*

Proposition 16 (Utilisation de la fermeture par complémentation) *Si $\overline{L_?} = L_{\text{nonreg}}$, alors $L_?$ est non régulier. Si $\overline{L_?} = L_{\text{reg}}$, alors $L_?$ est régulier.*

Dans les deux propositions suivantes, nous supposons que $L_?$ et L_{nonreg} sont deux langages sur Σ et Σ' , respectivement. De plus, nous considérons un morphisme f de Σ vers Σ'^* .

Proposition 17 (Utilisation de la fermeture par morphisme) *Supposons que $L_{\text{nonreg}} = f(L_?)$. Alors $L_?$ est non régulier. (En effet, si $L_?$ était régulier, comme les langages réguliers sont fermés par morphisme et $L_{\text{nonreg}} = f(L_?)$, alors L_{nonreg} serait régulier.)*

Proposition 18 (Utilisation de la fermeture par morphisme inverse) *Supposons que $L_{\text{nonreg}} = f^{-1}(L_?)$. Alors $L_?$ est non régulier. (En effet, si $L_?$ était régulier, comme les langages réguliers sont fermés par morphisme inverse et $L_{\text{nonreg}} = f^{-1}(L_?)$, alors L_{nonreg} serait régulier.)*

14.2.3 Une condition suffisante pour la non-régularité

Soit L un langage sur un alphabet Σ .

Proposition 19 (Condition suffisante pour la non-régularité d'un langage) *S'il existe deux suites de mots $(\alpha_n)_{n \in \mathbb{N}}$ et $(\beta_n)_{n \in \mathbb{N}}$ telles que :*

$$\forall i, j \in \mathbb{N}, \alpha_i \cdot \beta_j \in L \iff i = j,$$

alors L n'est pas régulier.

14.3 Exercices

14.3.1 Lemme de l'itération

Dans cette section, nous considérons les alphabets $\Sigma_1 = \{a, b\}$ et $\Sigma_2 = \{a, b, c\}$. Dans les exercices suivants, lorsqu'il faut montrer que les langages proposés ne sont pas réguliers, il faut utiliser le lemme de l'itération (lemme 6) en faisant des preuves par l'absurde, comme décrit dans la section 14.2.1.

Exercice 151 (♠) — Utilisation du lemme de l'itération

Démontrer que les langages suivants ne sont pas réguliers.

- | | |
|-------------------------------------------------------------------|------------------------------------------------------------------------------|
| 1. $\{a^i b^i \in \Sigma_1^* \mid i \geq 0\}.$ | 6. $\{a^{2 \times i} b^{2 \times i} \in \Sigma_1^* \mid i \in \mathbb{N}\}.$ |
| 2. $\{ww \mid w \in \Sigma_2^*\}.$ | 7. $\{a^i b^j c^{i+j} \in \Sigma_2^* \mid i, j \in \mathbb{N}\}.$ |
| 3. $\{a^i b^j \in \Sigma_1^* \mid i > j\}.$ | 8. $\{a^n b a^n \in \Sigma_2^* \mid n \in \mathbb{N}\}.$ |
| 4. $\{a^n b^{n+1} \in \Sigma_1^* \mid n \in \mathbb{N}\}.$ | 9. $\{a^i b^j \in \Sigma_1^* \mid i < j\}.$ |
| 5. $\{a^n b^{2 \times n} \in \Sigma_1^* \mid n \in \mathbb{N}\}.$ | 10. $\{a^n b^n c^n \in \Sigma_2^* \mid n \in \mathbb{N}\}.$ |
| | 11. $\{a^n b a^m b a^{n+m} \in \Sigma_1^* \mid n, m \in \mathbb{N}\}.$ |

Exercice 152 (♠♦) — Utilisation du lemme de l'itération

Démontrer que les langages suivants ne sont pas réguliers.

- | | |
|-------------------------------------------------------|---------------------------------------------------------------------------------|
| 1. $\{w \in \Sigma_2 \mid w _a = w _b\}.$ | 3. $\{a^{2 \times i} \cdot (b \cdot c)^i \in \Sigma_2 \mid i \in \mathbb{N}\}.$ |
| 2. $\{w \in \Sigma_2^* \mid w _a = w _b + w _c\}.$ | 4. $\{w \cdot w \cdot w \in \Sigma_2^* \mid w \in \Sigma_2^*\}.$ |

Exercice 153 (♠♦♣) — Utilisation du lemme de l'itération

Démontrer que les langages suivants ne sont pas réguliers.

- | | |
|-----------------------------------------------------------------|----------------------------------------------------------------------------------------|
| 1. $\{a^i \mid i \in \Sigma_1^* \text{ est un carré}\}.$ | 4. $\{a^i \in \Sigma_1^* \mid i \text{ est premier}\}.$ |
| 2. $\{a^i \in \Sigma_1^* \mid i \text{ est un cube}\}.$ | 5. $\{w \in \Sigma_2^* \mid \frac{ w _a}{ w _b} \in \mathbb{N}\}.$ |
| 3. $\{a^i \in \Sigma_1^* \mid i \text{ est une factorielle}\}.$ | 6. $\{w \in \Sigma_2^* \mid \frac{ w _a}{ w _b} \in \mathbb{N} \text{ est premier}\}.$ |

Exercice 154 (♠♦♣♥) — Utilisation du lemme de l'itération

Démontrer que les langages suivants ne sont pas réguliers.

1. $\{a^i b^j \in \Sigma_1^* \mid i \neq j\}$.
2. $\{a^m b a^n \in \Sigma_1^* \mid m, n \in \mathbb{N} \wedge m \neq n\}$.
3. $\{a^i b^j \in \Sigma_1^* \mid i \text{ et } j \text{ sont premiers entre eux}\}$.

Exercice 155 (♠♦) — Régulier ou non régulier ?

Dire si les langages suivants sur Σ_2 sont réguliers ou non. Justifier votre réponse en donnant un automate reconnaisseur lorsque le langage est régulier ou en utilisant le lemme de l'itération pour démontrer que le langage est non régulier.

- | | |
|-----------------------------------------------------------|---------------------------------------------------------------------------------------|
| 1. $\{a^i a^j \in \Sigma_2^* \mid i \neq j\}$. | 4. $\{w \cdot w^R \in \Sigma_2^* \mid w \in \Sigma_2^*\}$. |
| 2. $\{a^n c b^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$. | 5. $\{w \cdot u \cdot w^R \in \Sigma_2^* \mid w \in \Sigma_2^*, u \in \Sigma_2^+\}$. |
| 3. $\{a^n a^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$. | |

Exercice 156 (♠♦♣) — Un langage non régulier qui satisfait le lemme de l'itération

Soit X un langage non régulier sur l'alphabet $\Sigma = \{a, b\}$. Nous considérons les langages suivants sur Σ :

- $A = \{aba\} \cdot \Sigma^*$
- $B = \overline{A}$
- $C = (\{aba\} \cdot X) \cup B$

1. Démontrer que C satisfait le lemme de l'itération.
2. Démontrer que C est non régulier.

14.3.2 Fermeture des langages réguliers**Exercice 157 (♠) — Utilisation de la fermeture par complémentation**

Dans cet exercice, il est possible d'utiliser le fait que certains langages ont été démontrés comme non réguliers.

1. Démontrer que $\{a^p \mid p \text{ non premier}\}$ est non régulier.
2. Démontrer que $\{w \mid |w|_a \neq |w|_b\}$ est non régulier.

Exercice 158 (♠) — Utilisation de la fermeture par intersection

Supposons que nous avons démontré que $\{a^k \cdot b^k \mid k \in \mathbb{N}\}$ est non régulier.

1. Démontrer que $\{w \mid |w|_a = |w|_b\}$ est non régulier.

Exercice 159 (♠) — Utilisation de la fermeture par morphisme

1. Démontrer que $\{0^k 1^k 2^k \mid k \in \mathbb{N}\}$ est non régulier.
2. Démontrer que $\{a^n cb^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$ est non régulier.

Exercice 160 (♠♠♠) — Correct ou incorrect

Pour chacune des propositions suivantes, dire si elle est correcte ou non. Si elle est correcte, donner une preuve. Si elle est incorrecte, donner un contre-exemple.

1. Si $A \cup B$ est régulier et A est régulier, alors B est régulier.
2. Si $A \cap B$ est régulier et A est régulier, alors B est régulier.
3. Si $A \cup B$ est non régulier et A est non régulier, alors B est non régulier.
4. Si $A \cap B$ est non régulier et A est non régulier, alors B est non régulier.
5. Si $A \cup B$ est non régulier et A est régulier, alors B est non régulier.
6. Si $A \cap B$ est non régulier et A est régulier, alors B est non régulier.
7. Si A est régulier et B est non régulier, alors $A \cup B$ est non régulier.
8. Si A est régulier et B est non régulier, alors $A \cap B$ est non régulier.

Exercice 161 (♠♠♠) — Utilisation des propriétés de fermeture

Considérons l'alphabet $\Sigma = \{a, b\}$. En utilisant les résultats de la section 14.3.1, démontrer que les langages suivants sont non réguliers.

1. $\{a^m b^k d^n \mid m, k, n \in \mathbb{N} \wedge m \neq n\}.$
2. $\{w \in \Sigma^* \mid w \neq w^R\}.$

Exercice 162 (♠♠) — Utilisation des propriétés de fermeture

Considérons un alphabet Σ tel que $\Sigma \supseteq \{a, b, c\}$. Dans cet exercice, il faut utiliser les propriétés de fermeture des langages réguliers pour montrer que les langages suivants ne sont pas réguliers.

1. Démontrer que $\{u \in \Sigma^* \mid |u| \text{ est premier}\}.$
2. Démontrer que $\{u \in \Sigma^* \mid |u| \text{ est un carré}\}.$
3. Démontrer que $\{a^i b^j c^{i+j} \mid i, j \in \mathbb{N}\}.$

Exercice 163 (♠♠♠) — Lien entre deux langages

Nous considérons l'alphabet $\Sigma = \{a, b\}$ et les langages :

- $L_1 = \{u \cdot v \cdot w \mid u, w \in \Sigma^* \wedge v \in \{aaa, bbb\}\}$, et
- $L_2 = \{(aab)^n \cdot (abb)^n \mid n \in \mathbb{N}\}.$

1. Est-ce que le langage L_1 est un langage à états ? Si oui, donner un automate reconnaissant L_1 .
2. Remarquer une propriété sur la longueur des mots de L_2 .
3. Donner les facteurs n'apparaissant pas dans L_2 .

4. Déduire de la question précédente une relation entre L_1 et L_2 .
5. Est-ce que le langage L_2 est un langage à états ? Donner une preuve.
6. Est-ce que le langage $L_1 \cup L_2$ est un langage à états ? Donner une preuve.

14.3.3 Une condition suffisante pour la non-régularité

Dans ces exercices, il faut utiliser la condition suffisante pour montrer la non-régularité (propriété 19).

Exercice 164 (♠) — Utilisation de la condition suffisante

Considérons l'alphabet $\Sigma = \{a, b\}$.

1. Démontrer que le langage $\{a^k b^k \mid k \in \mathbb{N}\}$ est non régulier.

Exercice 165 (♠♦) — Utilisation de la condition suffisante

Démontrer que les langages suivants ne sont pas réguliers.

1. $\{w \in \Sigma^* \mid w \text{ est un palindrome}\}$. Pour rappel, un mot w est un palindrome si $w = w^R$, c'est-à-dire si le mot est égal à son mot miroir.
2. $\{w \cdot w \mid w \in \Sigma^*\}$.

Exercice 166 (♠♦♦) — Utilisation de la condition suffisante

Considérons un alphabet Σ .

1. Démontrer que $\{u \cdot a^{|u|} \mid u \in \Sigma^*\}$ n'est pas un langage régulier.

Exercice 167 (♠♦♦) — Démonstration de la condition suffisante

1. Démontrer la condition suffisante de non-régularité.

14.4 Indications pour résoudre les exercices

Indications pour l'exercice 153 (p. 283)

Utiliser les propriétés arithmétiques du carré, du cube et de la divisibilité des nombres premiers pour trouver le facteur itérant.

Indications pour l'exercice 154 (p. 284)

Utiliser les propriétés arithmétiques de la factorielle pour trouver le facteur itérant.

Indications pour l'exercice 156 (p. 284)

1. La constante d'itération de ce langage est 4.
2. Utiliser la fermeture des langages réguliers.

Indications pour l'exercice 160 (p. 285)

Certaines propriétés sont fausses, les autres découlent des propriétés de fermeture.

Indications pour l'exercice 161 (p. 285)

1. Utiliser l'expression régulière $a^* \cdot b \cdot a^*$.
2. Utiliser la fermeture par intersection.

Indications pour l'exercice 163 (p. 285)

5. Utiliser la condition suffisante à la non-régularité.

Indications pour l'exercice 167 (p. 286)

1. Faire une preuve par l'absurde en supposant le langage régulier. Utiliser l'automate reconnaisseur et le principe des tiroirs.

14.5 Solutions des exercices

14.5.1 Lemme de l'itération

Pour résoudre les exercices de cette section, pour chaque question, nous utilisons le lemme de l'itération sur le langage de la question que nous appelons L . Par ailleurs, les preuves sont similaires et diffèrent uniquement par : 1) le choix du mot w (dans le langage et de longueur plus grande que la constante d'itération), 2) le choix du facteur itérant k , 3) le raisonnement permettant d'obtenir une contradiction.

Chaque preuve suit les étapes suivantes :

- Supposer que le langage est régulier et satisfait le lemme de l'itération.
- Choisir un mot w dans le langage et de longueur plus grande qu'une de ses constantes d'itération N (non connues et fournies par le lemme).
- Déduire du lemme l'existence d'une décomposition du mot qui satisfait les propriétés données par le lemme.
- Exploiter les propriétés sur la décomposition pour déduire des informations sur le mot choisi (et sa structure).
- Choisir une valeur pour le facteur itérant k qui permet d'arriver à une contradiction.

Solution de l'exercice 151 (page 283)

Pour les premières questions, nous donnons l'énoncé complet de la preuve. Pour les questions suivantes, nous énonçons les éléments de la preuve spécifiques aux langages traités, comme décrit dans le préambule de cette section.

1. Supposons que $L = \{a^i b^i \in \Sigma_1^* \mid i \geq 0\}$ soit régulier, il satisfait donc le lemme de l'itération. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons le mot $w = a^N b^N$. On a $w \in L$ et $|w| \geq N$. Soient $x, y, z \in \Sigma^*$ des mots formant

une décomposition de w comme donnée par le lemme. Alors, comme $|xy| \leq n$ et $y \neq \epsilon$, on a $x = a^j$ et $y = a^i$ avec $j \geq 0$ et $i > 0$. Considérons le mot $w' = xy^2z = a^j a^i a^i a^{N-j-i} b^N = a^{N+i} b^N$. Alors, d'une part on devrait avoir (d'après le lemme) $w' \in L$ et d'autre part $w' \notin L$ car $N + i > N$. Ceci est une contradiction. Donc L n'est pas régulier.

2. Supposons que $L = \{ww \mid w \in \Sigma_2^*\}$ soit régulier, il satisfait donc le lemme de l'itération. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons le mot $w = a^N ba^N b$. On a $w \in L$ et $|w| \geq N$. Soient $x, y, z \in \Sigma^*$ des mots formant une décomposition de w comme donnée par le lemme. Alors, comme $|xy| \leq N$ et $y \neq \epsilon$, on a $x = a^j$ et $y = a^i$ avec $j \geq 0$ et $i > 0$. Soit $w' = xy^2z = a^j a^{2\times i} a^{N-i-j} ba^N b = a^{N+i} ba^N b$. Alors, d'une part on devrait avoir (d'après le lemme) $w' \in L$ et d'autre part $w' \notin L$ car il n'existe pas de w'' tel que $w' = w''w''$. Ceci est une contradiction. Donc L n'est pas régulier.
3. Supposons que $L = \{a^i b^j \in \Sigma_1^* \mid i > j\}$ soit régulier, il satisfait donc le lemme de l'itération. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons le mot $w = a^{N+1} b^N$. On a $w \in L$ car $N + 1 > N$ et $|w| \geq N$. Soient $x, y, z \in \Sigma^*$ des mots formant une décomposition de w comme donnée par le lemme. Alors, comme $|xy| \leq N$ et $y \neq \epsilon$, on a $x = a^j$ et $y = a^i$ avec $j \geq 0$ et $i > 0$. On a $w = a^j a^i a^{N+1-i-j} b^N$ avec $j \geq 0$. Soit $w' = xy^0 z = a^{N+1-i} b^N$. Alors, d'une part on devrait avoir $w' \in L$ et d'autre part $w' \notin L$ car $|w'|_a \leq |w'|_b$ ($i \geq 1$). Ceci est une contradiction. Donc L n'est pas régulier.
4. Supposons que $L = \{a^n b^{n+1} \in \Sigma_1^* \mid n \in \mathbb{N}\}$ soit régulier, il satisfait donc le lemme de l'itération. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons le mot $w = a^N \cdot b^{N+1}$. On a $w \in L$ et $|w| \geq N$. Soient $x, y, z \in \Sigma^*$ des mots formant une décomposition de w comme donnée ci-dessus par le lemme. Comme $|x \cdot y| \leq N$, x et y ne contiennent que des occurrences du symbole a : $\exists i, j \in \mathbb{N}. x = a^i \wedge y = a^j$. De plus, $y \neq \epsilon$ nous donne $j \neq 0$. De plus, $x \cdot y^0 \cdot z = a^{N-j} \cdot b^{N+1} \in L$. Or, ceci est impossible car $(N - j) + 1 \neq N + 1$ car $i \neq 0$. Ceci est une contradiction. Donc L n'est pas régulier.
5. Pour montrer que $\{a^n b^{2 \times n} \in \Sigma_1^* \mid n \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b^{2 \times N}$ et n'importe quel exposant du facteur itérant $k \neq 1$.
6. Pour montrer que $\{a^{2 \times i} b^{2 \times i} \in \Sigma_1^* \mid i \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^{2 \times N} \cdot b^{2 \times N}$ et n'importe quel exposant du facteur itérant $k \neq 1$.
7. Pour montrer que $\{a^i b^j c^{i+j} \in \Sigma_2^* \mid i, j \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b^N \cdot c^{2 \times N}$ et n'importe quel exposant du facteur itérant $k \neq 1$.
8. Pour montrer que $\{a^n b a^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b \cdot a^N$ et n'importe quel exposant du facteur itérant $k \neq 1$.
9. Pour montrer que $\{a^i b^j \in \Sigma_1^* \mid i < j\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b^{N+1}$ et n'importe quel exposant du facteur itérant $k > 1$.
10. Pour montrer que $\{a^n b^n c^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b^N \cdot a^N$ et n'importe quel exposant du facteur itérant $k \neq 1$.
11. Pour montrer que $\{a^n b^m c^{n+m} \in \Sigma_1^* \mid n, m \in \mathbb{N}\}$ n'est pas régulier, nous prenons le mot $w = a^N \cdot b a^N \cdot a^{2 \times N}$ et n'importe quel exposant du facteur itérant $k \neq 1$.

Solution de l'exercice 152 (page 283)

Nous dénotons par N une constante d'itération du langage concerné. Nous détaillons la première première preuve et, pour les questions suivantes, nous indiquons le mot à considérer et l'exposant du facteur itérant d'une décomposition donnée par le lemme.

1. Supposons que $L = \{w \in \Sigma_2^* \mid |w|_a = |w|_b\}$ soit régulier. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons le mot $w = a^N b^N$. On a $w \in L$ car $|w|_a = |w|_b$ et $|w| \geq N$. Soient $x, y, z \in \Sigma^*$ des mots formant une décomposition de w comme donnée par le lemme. Alors, comme $|xy| \leq N$ et $y \neq \epsilon$, on a $x = a^j$ et $y = a^i$ avec $j \geq 0$ et $i > 0$. Considérons le mot $w' = xy^2z = a^j a^i a^i a^{N-j-i} b^N = a^{N+i} b^N$. Alors, d'un côté on a $w' \in L$ mais aussi $w' \notin L$ car $N + i > N$. Ceci est une contradiction. Donc L n'est pas régulier.
2. Supposons que $\{w \in \Sigma_2^* \mid |w|_a = |w|_b + |w|_c\}$ soit régulier. Considérons le mot $w = a^{2N} b^N c^N$ ou aussi le mot $w = a^N b^N$ (de la réponse précédente). Pour chacun de ces mots, nous pouvons prendre n'importe quel exposant du facteur itérant $k \neq 1$.
3. Supposons que $\{a^{2 \times i} \cdot (b \cdot c)^i \in \Sigma_2^* \mid i \in \mathbb{N}\}$ soit régulier. Considérons le mot $w = a^{2 \times N} (b \cdot c)^N$. Nous prenons n'importe quel exposant du facteur itérant $k \neq 1$.
4. Supposons que $\{w \cdot w \cdot w \in \Sigma_2^* \mid w \in \Sigma_2^*\}$ soit régulier. Considérons le mot $w = a^N \cdot b \cdot a^N \cdot b \cdot a^N \cdot b$. Nous prenons n'importe quel exposant du facteur itérant $k \neq 1$.

Solution de l'exercice 153 (page 283)

1. Supposons que $\{a^i \mid i \in \Sigma_1^*\}$ soit régulier. Nous prenons le mot $w = a^{N^2}$ et une décomposition en x, y, z comme donnée par le lemme. Soit $m = |y|$. Comme $|xy| \leq N$, $m \leq N$. Nous prenons l'exposant du facteur itérant $k = 2$. Nous obtenons le mot $w' = a^{N^2+m}$ qui ne peut pas appartenir au langage. En effet, nous avons $N^2 + m \leq N^2 + N < (N+1)^2$ (car $|xy| \leq N$).
2. Supposons que $\{a^i \in \Sigma_1^* \mid i \text{ est un cube}\}$ soit régulier. Nous prenons le mot $w = a^{N^3}$ et une décomposition en x, y, z comme donnée par le lemme. Soit $m = |y|$. Comme $|xy| \leq N$, $m \leq N$. Nous prenons l'exposant du facteur itérant $k = 2$. En effet, nous avons $N^3 + m \leq N^3 + N < (N+1)^3$ (car $|xy| \leq N$).
3. Supposons que $\{a^i \in \Sigma_1^* \mid i \text{ est une factorielle}\}$ soit régulier. Nous prenons le mot $w = a^{N!}$. Nous prenons l'exposant du facteur itérant $k = N! + 1 - \frac{N!}{i}$ où i est le nombre de a dans y . Nous obtenons le mot $w' = a^{N!+i}$, comme $0 < i \leq N < N+1$, alors $N! < N! \times i < (N+1)!$.
4. Supposons que $\{a^i \in \Sigma_1^* \mid i \text{ est premier}\}$ soit régulier. Soit $L = \{a^i \mid i \text{ est premier}\}$, nous souhaitons démontrer que L n'est pas régulier. Supposons que L est régulier, il satisfait donc le lemme de l'itération. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Considérons un nombre premier p tel que $p \geq N$. Un tel nombre existe car il existe une infinité de nombres premiers. Considérons le mot $w = a^p$, nous avons $|w| \geq N$. Il existe $x, y \in \mathcal{L}_{\text{ER}}(a^*)$, $0 < m \leq p$ tel que $w = xa^m z$ et $|x| + |z| + m = p$. Considérons $xa^{m \times (p+1)} z$. D'une part, $xa^{m \times (p+1)} z \in L$. D'autre part, $|xa^{m \times (p+1)} z| = |x| + |z| + m \times (p+1) = |x| + |z| + m \times (p+1) = |x| + |z| + m + m \times p = (m+1) \times p$. Ceci est une contradiction.

5. Supposons que $\{w \in \Sigma_2^* \mid \frac{|w|_a}{|w|_b} \in \mathbb{N}\}$ soit régulier. Nous pouvons prendre le mot $w = a^N \cdot b^N$ et l'exposant du facteur itérant $k = 0$. Alternativement, nous pouvons prendre le mot $w = b^N \cdot a^N$ et l'exposant du facteur itérant $k > 1$.
6. Supposons que $\{w \in \Sigma_2^* \mid \frac{|w|_a}{|w|_b} \in \mathbb{N} \text{ est premier}\}$ soit régulier. Nous pouvons prendre le mot $w = a^p \cdot b$ avec $p \geq N$. Nous prenons l'exposant du facteur itérant $k = p + 1$ et obtenons le mot $w' = a^{p \times (i+1)} \cdot b$ qui nous donne la contradiction.

Solution de l'exercice 154 (page 284)

1. Supposons que $L = \{a^i b^j \mid i \neq j\}$ soit régulier. Alors, soit N une constante d'itération de ce langage. Prenons le mot $w = a^N b^{N!+N}$. Le mot w est de longueur $N! + 2N \geq N$ et $w \in L$. En utilisant le lemme de l'itération, w peut s'écrire xyz avec $|xy| \leq N$ et $y = a^l$ avec $0 < l \leq N$. Prenons l'exposant du facteur itérant $k = \frac{N!}{l} + 1$. Comme l divise $N!$, $k \in \mathbb{N}$. En utilisant le lemme de l'itération, nous avons $xy^k z \in L$. Le nombre de a dans $xy^k z$ est $N - l + k \times l = N - l + l \times (\frac{N!}{l} + 1) = N - l + N! + l = N + N!$. Ainsi, $xy^k z = a^{N!+N} b^{N!+N} \notin L$. Ceci est une contradiction.
2. Supposons que $\{a^m b a^n \in \Sigma_1^* \mid m, n \in \mathbb{N} \wedge m \neq n\}$ soit régulier. De manière similaire à la réponse précédente, nous utilisons le lemme de l'itération. Soit $N \geq 0$ une constante d'itération de ce langage. Nous considérons le mot $w = a^N b a^N$ et le même facteur itérant pour trouver une contradiction.
3. Supposons que $L = \{a^i b^j \mid i$ et j sont premiers entre eux $\}$ soit régulier. Alors, soit $N \geq 0$ une constante d'itération de ce langage. Prenons le mot $w = a^{N!} b^{N!+1}$. Le mot w est de longueur $2 \times N! + 1$ et $w \in L$. En utilisant le lemme de l'itération, w peut s'écrire xyz avec $|xy| \leq N$ et $y = a^l$ avec $0 < l \leq N$. Prenons l'exposant du facteur itérant $k = N! + \frac{N!}{l} + 2$. Nous avons $k \in \mathbb{N}$ comme l divise $N!$ et $N! + \frac{N!}{l} \geq 2$. En utilisant le lemme de l'itération, nous avons $xy^k z \in L$. Cependant :

$$xy^k z = a^{N!-l} + a^{l*(N!+\frac{N!}{l}+2)} b^{N!+1} = a^{N!-l} a^{(l+1)*N!+2 \times l} b^{N!+1}$$

Ainsi, $w' = xy^k z \notin L$. Ceci est une contradiction.

Solution de l'exercice 155 (page 284)

1. Le langage $\{a^i a^j \in \Sigma_2^* \mid i \neq j\}$ est égal au langage $\{a^i \mid i \geq 1\}$ qui est l'ensemble des mots de longueur supérieure ou égale à 1 formés par concaténation du symbole a . Ce langage est régulier.
2. Le langage $\{a^n c b^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$ est non régulier. Nous pouvons utiliser le lemme de l'itération en considérant le mot $w = a^N c b^N$ (où N est une constante d'itération). De plus, nous prenons $k = 0$ pour l'exposant du facteur d'itération.
3. Le langage $\{a^n a^n \in \Sigma_2^* \mid n \in \mathbb{N}\}$ est égal au langage $\{a^{2 \times n} \mid n \in \mathbb{N}\}$ qui est l'ensemble des mots de longueur paire formés par une concaténation du symbole a . Ce langage est régulier et est reconnu par l'AFED représenté dans la figure 3.4a (p. 59).
4. Le langage $\{w \cdot w^R \in \Sigma_2^* \mid w \in \Sigma_2^*\}$ est non régulier. Nous pouvons utiliser le lemme de l'itération en considérant le mot $w = a^N b b a^N$ (où N est une constante d'itération). De plus, nous prenons $k = 0$ pour l'exposant du facteur itérant.

5. Le langage $\{w \cdot u \cdot w^R \in \Sigma_2^* \mid w \in \Sigma_2^*, u \in \Sigma_2^+\}$ est non régulier. Nous pouvons utiliser le lemme de l'itération en considérant le mot $w = a^N ba^N$ (où N est une constante d'itération). De plus, nous prenons $k = 0$ pour l'exposant du facteur itérant.

Solution de l'exercice 156 (page 284)

- Le langage C satisfait effectivement le lemme de l'itération et sa constante d'itération est 4. Soit w un mot dans le langage et longueur supérieure ou égale à 4. Nous distinguons plusieurs cas en fonction du préfixe de w .

- Considérons le cas où aba est un préfixe de w . Nous pouvons décomposer $w = xyz$ comme suit $x = \epsilon, y = a, z = ba.w'$ où w' est le suffixe de w obtenu en supprimant le préfixe aba de w . Considérons les mots w_k de la forme $xy^k z = a^k b a w'$, pour $k \geq 0$. Nous avons bien $|xy| \leq 4$ et $y \neq \epsilon$.
 - Pour $k = 0$, $w_0 = ba \cdot w'$ qui n'appartient pas à A (car il ne commence pas par le préfixe aba) et appartient donc à B et à C .
 - Pour $k = 1$, $w_1 = w$ qui appartient à C .
 - Pour $k > 1$, w_k a comme préfixe aa (et ne peut donc pas avoir aba comme préfixe) et appartient à C .
- Considérons le cas où aba n'est pas un préfixe de w . Soit p le préfixe de longueur 3 de w , $p = w_{\dots 3}$. Nous pouvons décomposer $w = xyz$ comme suit $x = p, y = w(4), z = w_{5\dots}$ où $w(4)$ est la quatrième lettre de w et $w_{5\dots}$ est le suffixe de w à partir de sa cinquième lettre (possiblement vide). Considérons les mots w_k de la forme $xy^k z$, pour $k \geq 0$. Nous avons bien $|xy| \leq 4$ et $y \neq \epsilon$. Pour tout $k \geq 0$, aba n'est pas un préfixe de w_k et ainsi $w_k \in L$.

Nous avons montré que C satisfait le lemme de l'itération et que 4 est une constante d'itération de ce langage.

- Nous savons que A est un langage régulier. Pour montrer que C est non régulier, nous montrons que son intersection avec un langage régulier donne un langage non régulier (propriété 15). Nous utilisons ainsi la fermeture des langages régulier et en particulier le fait que l'intersection de deux langages réguliers est un langage régulier. En particulier, $C \cap A = \{aba\} \cdot X \cap A \cup B \cap A = \{aba\} \cdot X$ qui est non régulier.

14.5.2 Fermeture des langages réguliers

Solution de l'exercice 157 (page 284)

Les deux preuves utilisent directement la fermeture par intersection (propriété 15) et le fait que les langages suivants ont été démontrés non réguliers.

- $\{a^p \mid p \text{ premier}\}$ (exercice 153).
- $\{w \mid |w|_a = |w|_b\}$ (exercice 152).

Solution de l'exercice 158 (page 284)

- Nous savons que le langage dénoté par l'expression régulière $a^* \cdot b^*$ est régulier. Alors, comme $\{w \in \{a, b\}^* \mid |w|_a = |w|_b\} \cap \mathcal{L}_{\text{ER}}(a^* \cdot b^*) = \{a^k \cdot b^k \mid k \in \mathbb{N}\}$, nous en déduisons que $\{w \mid |w|_a = |w|_b\}$ est non régulier.

Solution de l'exercice 159 (page 285)

- Soit L_1 le langage donné dans l'énoncé. Soit $L_2 = \{a^k b^k \mid k \in \mathbb{N}\}$ que nous avons démontré comme non régulier. Considérons le morphisme induit par $h = \{0 \mapsto a, 1 \mapsto b, 2 \mapsto \epsilon\}$. Nous avons $h(L_1) = L_2$. Ainsi, en utilisant la fermeture par morphisme, L_1 est non régulier.
- Soit L_1 le langage donné dans l'énoncé. Soit $L_2 = \{a^n b a^n \mid n \in \mathbb{N}\}$ que nous avons démontré comme non régulier. Considérons le morphisme induit par $h = \{a \mapsto a, c \mapsto b, b \mapsto a\}$. Nous avons $h(L_1) = L_2$. Ainsi, en utilisant la fermeture par morphisme, L_1 est non régulier.

Solution de l'exercice 160 (page 285)

Nous considérons un alphabet et Σ^* le langage universel sur l'alphabet Σ .

- Cette proposition est fausse. Prenons $A = \Sigma^*$ et B un langage non régulier. Le langage A est régulier. Nous avons $A \cup B = A$ donc $A \cup B$ est régulier. Ainsi, A et $A \cup B$ sont réguliers, mais B n'est pas régulier.
- Cette proposition est fausse. Prenons $A = \emptyset$ et B un langage non régulier. Le langage A est régulier. Nous avons $A \cap B = A$ donc $A \cap B$ est régulier. Ainsi, $A \cap B$ et A sont réguliers, mais B n'est pas régulier. Nous aurions pu prendre également le contre-exemple suivant $A = \{\epsilon\}$, $B = \{a^n b^n \mid n \in \mathbb{N}\}$.
- Cette proposition est fausse. Prenons $A = \{a^n b^n \mid n \in \mathbb{N}\}$ et $B = \{\epsilon\}$. Le langage A est non régulier et le langage B régulier. Nous avons $A \cup B = A$ donc $A \cup B$ est non régulier.
- Cette proposition est fausse. Prenons $A = \{a^n b^n \mid n \in \mathbb{N}\}$ et $B = \{\epsilon\}$. Le langage A est non régulier et le langage B régulier. Nous avons $A \cap B = A$ donc $A \cap B$ est non régulier.
- Cette proposition est vraie. Si B était régulier, alors comme A est régulier, d'après la fermeture des langages réguliers par l'opérateur d'union ensembliste, $A \cup B$ serait régulier.
- Considérons la proposition "si $A \cap B$ est non régulier et A est régulier, alors B est non régulier". Cette proposition est vraie. Si B était régulier, alors comme A est régulier, d'après la fermeture des langages réguliers par l'opérateur d'union ensembliste, $A \cap B$ serait régulier.
- Cette proposition est fausse. Prenons $A = \Sigma^*$ et B un langage non régulier. Le langage A est régulier. Nous avons $A \cup B = A$ donc $A \cup B$ est régulier.
- Prenons $A = \emptyset$ et B un langage non régulier. Le langage A est régulier. Nous avons $A \cap B = A$ donc $A \cap B$ est régulier.

Solution de l'exercice 161 (page 285)

- Nous considérons le langage $\{a^n b a^m \mid n, m \in \mathbb{N} \wedge n \neq m\}$ que nous avons montré comme non régulier dans l'exercice 154 et le langage régulier décrit par l'expression régulière $a^* b a^*$. Nous avons :

$$\mathcal{L}_{\text{ER}}(a^* b a^*) \cap \{a^n b^k a^m \mid n, m, k \in \mathbb{N} \wedge n \neq m\} = \{a^n b a^m \mid n, m \in \mathbb{N} \wedge n \neq m\}.$$

Ainsi, comme le langage dénoté par l'expression régulière a^*ba^* est régulier et $\{a^nba^m \mid n,m \in \mathbb{N} \wedge n \neq m\}$ non régulier, le langage $\{a^n b^k a^m \mid n,m,k \in \mathbb{N} \wedge n \neq m\}$ ne peut pas être régulier.

2. Nous proposons deux solutions.

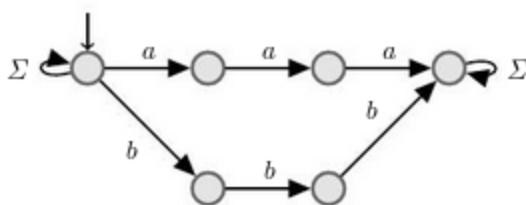
- La première solution utilise la fermeture des langages régulier par l'opérateur d'intersection ensembliste. Nous utilisons l'égalité entre langage suivante : $\{w \mid w = w^R\} \cap \{|w| = 2k, k \in \mathbb{N}\} = \{w \cdot w^R \mid w \in \Sigma^*\}$. Comme le langage $\{|w| = 2k, k \in \mathbb{N}\}$ est régulier et le langage $\{w \cdot w^R \mid w \in \Sigma^*\}$ non régulier, le langage $\{w \mid w = w^R\}$ ne peut pas être régulier.
 - La seconde solution utilise la fermeture des langages régulier par les opérateurs d'intersection et de complémentation ensembliste. D'abord, nous montrons que le langage $\{w \mid w = w^R\}$ est non régulier. Pour cela nous utilisons l'égalité entre langage suivante : $\{w \mid w = w^R\} \cap a^*ba^* = \{a^nba^n\}$. Comme le langage décrit par l'expression régulière a^*ba^* est régulier et que le langage $\{a^nba^n\}$ est non régulier (exercice 151), le langage $\{w \mid w = w^R\}$ ne peut pas être régulier. Ensuite, nous utilisons l'égalité entre langage suivante : $\{w \mid w \neq w^R\} = \{w \mid w = w^R\}$. Comme le langage $\{w \mid w = w^R\}$ est non régulier, le langage $\{w \mid w \neq w^R\}$ ne peut pas être régulier.

Solution de l'exercice 162 (page 285)

1. Soit $L = \{u \in \Sigma^* \mid |u| \text{ est premier}\}$. Considérons le morphisme induit par $h : \Sigma \rightarrow \{a\}$. Nous avons $h(L) = \{a^i \mid i \text{ est premier}\}$. En utilisant la (contraposée de la) propriété de fermeture des langages réguliers par morphisme, comme $h(L)$ n'est pas régulier, L ne l'est pas non plus.
 2. Soit $L = \{u \in \Sigma^* \mid |u| \text{ est un carré}\}$. Considérons le morphisme induit par $h : \Sigma \rightarrow \{a\}$. Nous avons $h(L) = \{a^i \mid i \text{ est un carré}\}$. En utilisant la (contraposée de la) propriété de fermeture des langages réguliers par morphisme, comme $h(L)$ n'est pas régulier, L ne l'est pas non plus.
 3. Soit $L = \{a^i b^j c^{i+j} \mid i, j \in \mathbb{N}\}$. Considérons le morphisme induit par $h : \{a, b, c\} \rightarrow \{a, b\}$ tel que $h(a) = a, h(b) = \epsilon, h(c) = b$, nous avons $h(L) = \{a^i b^i \mid i \in \mathbb{N}\}$. En utilisant la (contraposée de la) propriété de fermeture des langages réguliers par morphisme, comme $h(L)$ n'est pas régulier, L ne l'est pas non plus.

Solution de l'exercice 163 (page 285)

1. Ce langage est un langage à états. Nous donnons un automate reconnaisseur ci-dessous.



- Les mots de ce langage sont de longueur multiple de 6.
 - Les facteurs aaa et bbb n'apparaissent pas dans L_2 .

4. Nous avons l'égalité ensembliste suivante entre L_1 et L_2 : $L_1 \cap L_2 = \emptyset$.
5. Tous les mots de $\{(aab)^n \mid n \in \mathbb{N}\}$ sont distingués par L_2 . En effet, après chaque mot $\{(aab)^n \mid n \in \mathbb{N}\}$, il existe un suffixe unique tel que la concaténation de ce suffixe à un mot de $\{(aab)^n \mid n \in \mathbb{N}\}$ forme un mot de L_2 . Pour un mot $(aab)^i$ de $\{(aab)^n \mid n \in \mathbb{N}\}$, nous pouvons concaténer uniquement le mot $(aab)^i$. L_2 est donc un langage non régulier.
6. Nous savons que L_1 est régulier, ainsi est $\overline{L_1}$, et que, d'après la question précédente, L_2 est non régulier. Par ailleurs, nous avons $L_1 \cup L_2 \cap \overline{L_1} = L_2$. D'après la propriété de fermeture des langages réguliers, $L_1 \cup L_2$ est non régulier.

14.5.3 Une condition suffisante pour la non-régularité

Dans les trois exercices suivants, nous définissons des suites $(\alpha_n)_{n \in \mathbb{N}}$ et $(\beta_n)_{n \in \mathbb{N}}$. Nous avons que le mot $\alpha_i \cdot \beta_j$ appartient au langage considéré si et seulement si $i = j$. Ainsi, en utilisant la condition suffisante de non-régularité, nous obtenons le résultat attendu.

Solution de l'exercice 164 (page 286)

1. Pour montrer que le langage $\{d^k b^k \mid k \in \mathbb{N}\}$ est non régulier, nous considérons les suites $(\alpha_n)_{n \in \mathbb{N}} = a^n$ et $(\beta_n)_{n \in \mathbb{N}} = b^n$.

Solution de l'exercice 165 (page 286)

1. Pour montrer que le langage $\{w \in \Sigma^* \mid w \text{ est un palindrome}\}$ est non régulier, nous considérons les suites $(\alpha_n)_{n \in \mathbb{N}} = a^n$ et $(\beta_n)_{n \in \mathbb{N}} = b \cdot a^n$.
2. Pour montrer que le langage $\{w \cdot w \mid w \in \Sigma^*\}$ est non régulier, nous considérons les suites $(\alpha_n)_{n \in \mathbb{N}} = b \cdot a^n$ et $(\beta_n)_{n \in \mathbb{N}} = b \cdot a^n$.

Solution de l'exercice 166 (page 286)

1. Pour montrer que le langage $\{u \cdot a^{|u|} \mid u \in \Sigma^*\}$ est non régulier, nous considérons les suites $(\alpha_n)_{n \in \mathbb{N}} = b^n$ et $(\beta_n)_{n \in \mathbb{N}} = a^n$.

Solution de l'exercice 167 (page 286)

1. Nous démontrons la propriété par l'absurde. Soit L un langage. Supposons qu'il existe deux suites $(\alpha_n)_{n \in \mathbb{N}}$ et $(\beta_n)_{n \in \mathbb{N}}$ telles que $\forall i, j \in \mathbb{N}, \alpha_i \cdot \beta_j \in L \iff i = j$. Alors, nous avons : (1) $\forall i \in \mathbb{N}, \alpha_i \cdot \beta_i \in L$ et (2) $\forall i, j \in \mathbb{N}, i \neq j \implies \alpha_i \cdot \beta_j \notin L$. Supposons que L soit régulier. D'après le théorème de Kleene, il existe un AFED $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ qui reconnaît L . Soit $N = |Q|$. Considérons les mots α_i et β_i , pour $i \in [0, N]$. Considérons l'application de $[0, N]$ vers Q définie par $i \mapsto \delta^*(q_{\text{init}}, \alpha_i)$ (où δ^* est la fonction de transition étendue construite à partir de δ). D'après le principe des tiroirs, comme $|[0, N]| > |Q|$, il existe $q \in Q$ et $k, l \in [0, N]$ avec $k \neq l$ tels que $\delta^*(q_{\text{init}}, \alpha_k) = \delta^*(q_{\text{init}}, \alpha_l) = q$. En utilisant les mots β_k et β_l de la suite $(\beta_n)_{n \in \mathbb{N}}$, d'après (1), nous avons $\alpha_k \cdot \beta_k \in L$ et $\alpha_l \cdot \beta_l \in L$. Comme A reconnaît L , il existe $q_k, q_l \in F$ tels que $\delta^*(q_{\text{init}}, \alpha_k \cdot \beta_k) = q_k$ et $\delta^*(q_{\text{init}}, \alpha_l \cdot \beta_l) = q_l$. Comme A est déterministe, de $\delta^*(q_{\text{init}}, \alpha_k \cdot \beta_k) = q_k$ et $\delta^*(q_{\text{init}}, \alpha_k) = q$, nous déduisons $\delta^*(q, \beta_k) = q_k$. En utilisant $\delta^*(q_{\text{init}}, \alpha_l) = q$ et $\delta^*(q, \beta_k) = q_k$, nous déduisons $\delta^*(q_{\text{init}}, \alpha_l \cdot \beta_k) = q_k$. Comme $q_k \in F$, nous avons $\alpha_l \cdot \beta_k \in L$ (avec $l \neq k$). Ceci est une contradiction avec (2).

Annexe A

Modélisation et résolution de problèmes avec les automates

A.1 Exercices

Exercice 168 (♠♠♣) — Score au tennis

Au tennis les points sont comptés de la manière suivante : 15, 30, 40. Jusqu'à 40, les points sont comptés de façon incrémentale. Lorsqu'au moins un des deux joueurs atteint le score 40, si l'autre joueur a un score strictement inférieur et que le joueur à 40 marque un autre point, il remporte le jeu. Si les deux joueurs atteignent le score 40, alors le joueur remportant le point suivant obtient l'avantage. Si le joueur avec l'avantage marque un nouveau point il remporte le point. Sinon, si l'autre joueur marque un point, ils reviennent tous les deux au score de 40.

1. Donner un automate qui compte les points au tennis. On pourra utiliser l'état pour contenir le score et un alphabet à deux symboles, chaque symbole correspondant à la victoire d'un point.

Exercice 169 (♣♣♣) — Code pour le contrôle l'accès

Considérons l'alphabet formé par les chiffres utilisés en notation décimale : $\Sigma = \{0, \dots, 9\}$. Nous souhaitons modéliser des variantes d'automates qui permettent de reconnaître un code sous la forme $x \cdot y \cdot z$ avec $x, y, z \in \Sigma$. Pour chacune des questions suivantes, si cela est pertinent, considérer les deux alternatives suivant qu'il existe ou non un bouton « val » pour valider la saisie. Par exemple, il est possible de considérer un bouton pour valider sa saisie ou que chaque séquence de trois chiffres correspond à une saisie.

1. L'automate ne laisse qu'une seule chance.
2. L'automate ne laisse que deux chances. L'utilisateur entre dans sa deuxième chance lorsqu'il a saisi une séquence de 3 entrées différentes du code attendu.

3. L'automate ne laisse que deux chances. L'utilisateur entre dans sa deuxième chance dès qu'il saisit une entrée qui fait que sa première séquence de trois entrées ne plus correspondre au code attendu. Par exemple, après que l'utilisateur ait entré $x \cdot w$, la prochaine entrée saisie sera celle de sa seconde chance. Dans ce cas, le bouton de validation est inutile. On peut imaginer qu'un signal sonore est émis pour informer l'utilisateur que le code entré est incorrect.
4. L'automate ne laisse que deux chances. L'automate permet d'annuler sa saisie grâce à un bouton spécial.
5. L'automate accepte dès que les derniers chiffres entrés correspondent au code.

Exercice 170 (♠♠♠) — Machine à café

Nous souhaitons modéliser une machine à boissons simplifiée et son interaction avec des clients. La machine sert plusieurs types de boissons café, thé et chocolat. Toutes les boissons ont le même prix : 1 jeton. La machine doit laisser à l'utilisateur le choix de la boisson. Elle doit délivrer une boisson lorsque l'utilisateur a acquitté le prix de la boisson et effectué son choix. La machine doit laisser au client la possibilité de récupérer son jeton s'il n'y a pas encore confirmé son choix de boisson. L'utilisateur peut insérer un jeton, sélectionner une boisson, récupérer sa boisson, récupérer son jeton et demander l'annulation du service (si cela est possible).

Dans cet exercice, nous sommes en présence de deux automates, un modélisant un client, l'autre la machine à café. Pour faciliter la communication entre la machine et le client, nous considérons que les deux automates partagent le même alphabet et communiquent via le produit de deux automates. En conséquence de quoi, les entrées de la machine sont les sorties du client (par exemple, l'action de payer ou d'annuler) et les entrées du client sont les sorties de la machine (par exemple, récupérer la boisson ou un jeton).

1. Déterminer les actions communes de l'utilisateur et de la machine.
2. Modéliser le comportement de l'utilisateur à l'aide d'un automate.
3. Modéliser le comportement de la machine à l'aide d'un automate.
4. Comment obtenir le comportement global (observable) du système ? Décrire certains de ces comportements.
5. Nous souhaitons maintenant vérifier le modèle de la machine, en présence d'utilisateur. Supposons qu'une propriété soit représentée par un langage L_{prop} sur l'alphabet de la machine et de l'utilisateur. Nous considérons le langage L_{sys} des mots représentant les différentes exécutions de la machine en présence de l'utilisateur. Pour chacune des questions suivantes, donner une relation entre L_{prop} et L_{sys} qui est satisfaite si et seulement si la réponse à la question est affirmative.
 - Tous les comportements du système respectent la propriété.
 - Il est possible que le système réalise l'un des comportements de la propriétés.
 - Le système ne respecte pas la propriété.
6. Donner les algorithmes à utiliser pour répondre aux questions précédentes.
7. Afin de détecter les potentielles erreurs de modélisation et en utilisant les algorithmes vus en cours, déterminer comment vérifier les propriétés suivantes sur le comportement global du modèle :

- Est-ce que le client peut obtenir une boisson sans avoir inséré de jeton ?
- Est-ce que le client peut, après avoir inséré un jeton et choisi une boisson, ne pas obtenir de boisson (avant de commencer une nouvelle transaction) ?
- Est-ce que le système peut se bloquer ?

Exercice 171 (♠♠) — Une histoire de berger, de loup, de chèvre et de choux

Monsieur Berger B emmène un loup L , une chèvre C , et un chou X près d'une rivière et souhaite traverser avec un petit bateau. Le bateau est tellement petit que B peut entrer dans le bateau avec au plus un passager. Sans surveillance de B , L mange C et C mange X .

Nous souhaitons trouver comment B peut faire traverser la rivière à la compagnie sans perdre d'élément ?

1. Déterminer l'ensemble des états de l'automate qui représente les différentes situations des deux cotés de la rivière.
2. Donner un automate qui modélise la situation.
3. Utiliser l'automate trouvé à la question précédente pour trouver comment le problème peut être résolu de manière algorithmique.

Exercice 172 (♠♠♠) — Étrange planète

Nous considérons l'exemple fictif d'une étrange planète¹. Sur cette planète trois espèces cohabitent. Nous appellerons les espèces **rouge**, **bleue** et **orange**. Nous souhaitons modéliser l'évolution de la population de ces espèces selon leur mode de reproduction, qui suit les règles suivantes :

- 2 individus de deux *espèces différentes* peuvent s'unir ;
- la reproduction tue les deux individus ;
- la reproduction génère 2 individus de la troisième espèce.

Par exemple : 1 **rouge** et 1 **bleue** \rightarrow 2 **orange**. Nous supposons également que :

- des individus peuvent s'unir juste après avoir été générés (pas de distinction adulte/enfant) ;
- les individus ne peuvent mourir que lors de la reproduction.

1. Observer qu'avec les règles le nombre d'individus n'évolue pas et que la planète cesse son évolution si tous les individus sont de la même espèce.
2. La configuration de la planète est déterminée par le nombre d'individus de chaque espèce. Indiquer comment modéliser l'évolution de la population par un automate.
3. Modéliser l'évolution de la population sur des planètes avec 2, 3 et 4 individus.
4. Trouver une « symétrie » dans les états pour un nombre d'individus fixé.
5. Exprimer une condition pour l'arrêt de l'évolution sur l'ensemble des individus (et non la liste des individus).
6. Proposer une nouvelle représentation de l'état basée sur les observations faites dans les deux questions précédentes.

1. Inspiré d'un exemple du cours sur les automates à Stanford. Voir également [23, 14].

7. Revisiter les automates précédents pour une planète avec 2, 3 et 4 individus.
8. Proposer un automate pour une planète avec 5 individus.
9. Pour automate donné, un état courant donné, comment déterminer en raisonnant sur les chemins si :
 - a) l'évolution s'arrêtera inévitablement.
 - b) l'évolution ne peut pas s'arrêter.
 - c) l'évolution peut s'arrêter.

Exercice 173 (♠♠♣) — Récipients

Nous considérons la situation où nous disposons d'une arrivée d'eau (supposée infinie) et deux récipients de 3 et 5 litres, respectivement récipients A et B . Nous souhaitons utiliser un automate nous permettant de décrire comment obtenir précisément 4 litres dans le récipient de 5 litres. Il est uniquement possible de réaliser les actions suivantes :

- Compléter le contenu de n'importe quel récipient jusqu'à la contenance maximale (assurant ainsi que la quantité d'eau dans le récipient est égale à sa contenance).
- Transvaser le contenu d'un récipient vers l'autre récipient.
 - Transvaser le contenu de A vers B se fait de la manière suivante :
 - transférer intégralement en une fois le contenu de A si la capacité de B le permet ;
 - compléter le récipient B jusqu'à atteindre 5 litres avec une partie du contenu de A .
 - Transvaser le contenu de B vers A se fait de la manière suivante :
 - compléter le récipient A jusqu'à atteindre 3 litres avec une partie du contenu de B ;
 - transférer intégralement en une fois le contenu de B si la capacité de A le permet.
- Vider un des 2 récipients.

Ces opérations sont les seules permettant d'avoir une mesure précise de quantité d'eau dans les réservoirs. Nous remarquons que les changements de contenances se font en nombres entiers de litres.

1. Définir l'alphabet de l'automate.
2. Définir l'espace d'états de l'automate qui représente les différentes contenances des deux récipients.
3. Définir l'ensemble des états accepteurs de cet automate.
4. Définir l'alphabet de l'automate et la relation de transition entre états.
5. Supposons que cet automate soit construit. Comment résoudre le problème initial ?
6. Trouver un chemin partant de l'état initial menant à un état accepteur.

Exercice 174 (♠♠♣) — Opacité d'un système

Nous nous intéressons à une propriété importante en sécurité des systèmes informatiques : l'opacité². Le contexte est le suivant. Nous supposons qu'un attaquant observe un système

2. Voir également [6, 8].

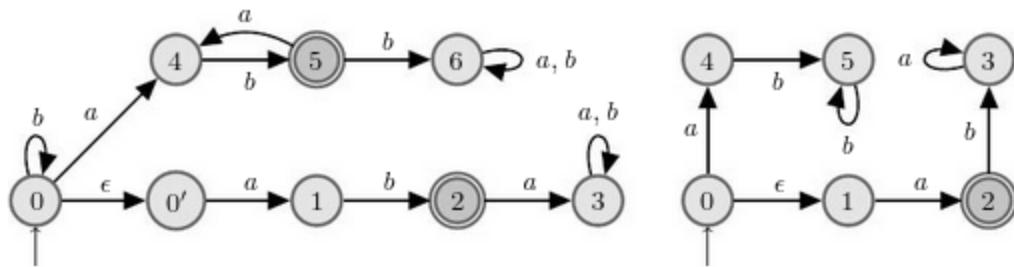


Figure A.1 – Automates modélisant des systèmes dont on veut étudier l'opacité (exercice 174).

dont le comportement est modélisé par un AFEND avec ϵ -transitions. Les états accepteurs de l'automate représentent le "secret" : lors d'une exécution du système, l'attaquant ne doit pas être en mesure de savoir *avec certitude* que le système est dans un état secret. Si lors d'une exécution du système, l'attaquant est en mesure de déterminer que le système est dans un état secret (sans savoir forcément précisément lequel), alors on dit que cette exécution *révèle le secret*. Un système est dit *opaque* s'il n'existe pas d'exécution qui révèle le secret. L'attaquant observe le système à travers une "fenêtre d'observation" qui lui permet de voir toutes les transitions exceptées les ϵ -transitions ; autrement dit, les ϵ -transitions sont non observables. Pour un automate $(Q, \Sigma, q_{\text{init}}, \Delta, F)$, un mot w est observé s'il existe une exécution $(q_{\text{init}}, w) \cdots (q, \epsilon)$, avec $q \in Q$. L'attaquant connaît l'automate.

1. Nous considérons le système représenté par l'automate dans la figure A.1a. Lorsque l'attaquant observe a , puis b et donc ab , quels sont les états courants possibles du système ?
 2. Dire si le système modélisé par l'automate représenté dans la figure A.1a est opaque.
 3. Dire si le système modélisé par l'automate représenté dans la figure A.1b est opaque.
 4. Pour un mot $w \in \Sigma^*$, donner une définition caractérisant le fait que w ne révèle pas le secret.
 5. En déduire une définition de l'opacité d'un système.
 6. Est-il possible à partir de l'automate modélisant un système, de construire un automate qui indique la connaissance de l'attaquant en fonction de son observation ?

A.2 Solutions des exercices

Solution de l'exercice 168 (page 295)

1. L'automate est représenté dans la figure A.2.

Solution de l'exercice 169 (page 295)

1. Dans le cas où chaque séquence de 3 chiffres consécutifs correspond à une entrée, nous obtenons l'automate représenté dans la figure A.3a. Dans le cas où il existe un bouton pour valider la saisie, nous obtenons l'automate représenté dans la figure A.3b.

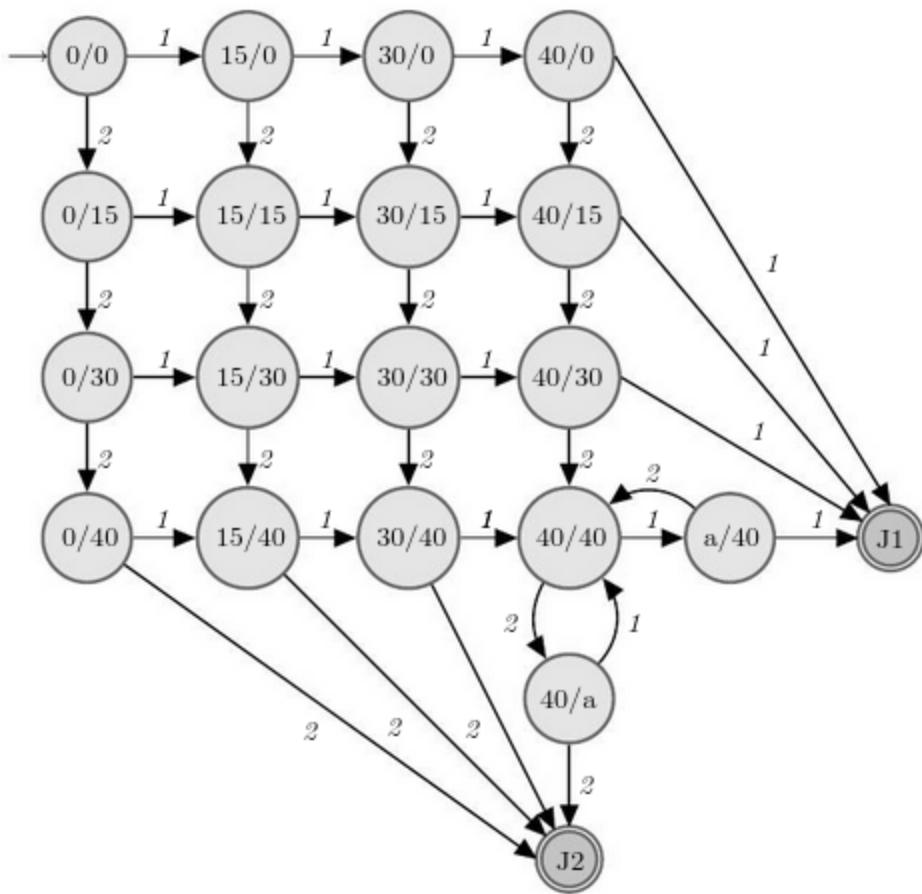


Figure A.2 – Automate solution de l'exercice 168.

Notons qu'avec l'automate de la figure A.3b, si l'utilisateur appui prématûrement sur le bouton de validation, alors le code entré est rejeté. Si on souhaitait autoriser l'appui prématûré sans conséquence sur le bouton de validation, il faudrait ajouter, sur chacun des états où il n'y a pas de transition sortante sur le symbole val, une transition de l'état sur lui même étiquetée par val.

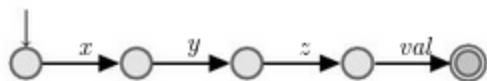
2. Dans le cas où chaque séquence de 3 chiffres consécutifs correspond à une entrée, et que l'automate ne laisse que deux chances, nous obtenons l'automate représenté dans la figure A.3c.

Dans le cas où il existe un bouton pour valider la saisie, nous obtenons l'automate représenté dans la figure A.3d. Nous faisons la même remarque qu'à la question précédente concernant l'appui prématûré sur le bouton de validation; ici l'appui prématûré supprime les deux chances.

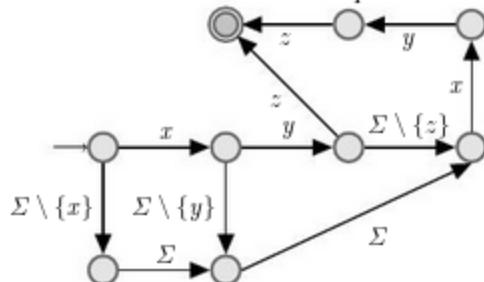
3. Dans le cas où l'automate ne laisse que deux chances et que la deuxième chance débute comme décrit dans l'énoncé, nous obtenons l'automate représenté dans la figure A.3e.



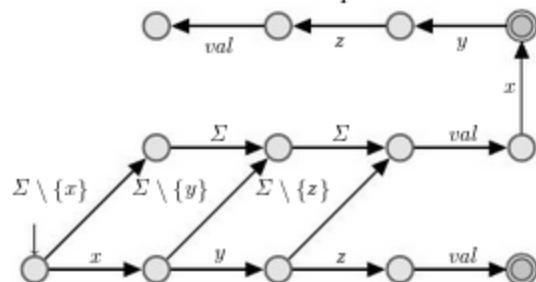
(a) Cas où chaque séquence de trois chiffres correspond à une entrée et l'automate ne laisse qu'une chance.



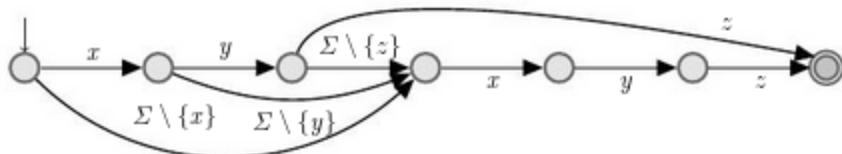
(b) Cas où il existe un bouton pour valider la saisie et l'automate ne laisse qu'une chance.



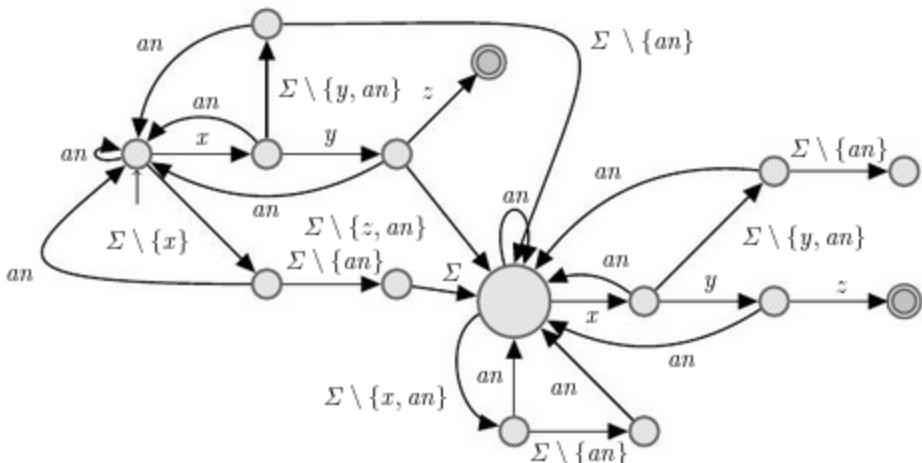
(c) Cas où chaque séquence de trois chiffres correspond à une entrée et l'automate laisse deux chances.



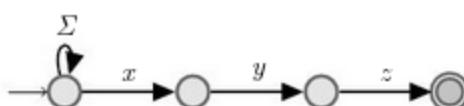
(d) Cas où il existe un bouton pour valider la saisie et l'automate laisse deux chances.



(e) Cas où l'automate laisse deux chances et la seconde entrée débute dès que la première entrée de l'utilisateur diffère du code attendu.



(f) Cas où chaque séquence de trois chiffres correspond à une entrée et l'automate dispose d'un bouton annuler (an).



(g) Cas où l'automate accepte dès que les trois derniers chiffres entrés sont corrects.

Figure A.3 – Automates pour l'exercice 169.

4. Dans le cas où l'automate ne laisse que deux chances et que l'automate permet d'annuler sa saisie, nous obtenons l'automate représenté dans la figure A.3f.
5. Dans le cas où l'automate accepte dès que les trois derniers chiffres sont corrects, nous obtenons l'automate non déterministe représenté dans la figure A.3g.

Solution de l'exercice 170 (page 296)

1. Les actions communes de l'utilisateur et de la machine sont dans l'alphabet :

$$\{ \text{paye}, \text{annule}, \text{choix_boisson}, \text{servir_boisson}, \text{recup_jeton} \} .$$

- L'action *paye* correspond à l'insertion par l'utilisateur d'un jeton.
 - L'action *annule* correspond à l'indication par l'utilisateur qu'il souhaite annuler la transaction (cela n'est pas forcément pris en compte par la machine).
 - L'action *choix_boisson* correspond à l'indication par l'utilisateur de la boisson désirée.
 - L'action *servir_boisson* correspond à la préparation et service par la machine de la boisson sélectionnée.
 - L'action *recup_jetton* correspond à la restitution du jeton par la machine.
2. Le comportement de l'utilisateur peut être modélisé par l'automate représenté dans la figure A.4a (p. 304). La transition en bleu permet d'autoriser plusieurs « sessions » de l'utilisateur, c'est-à-dire à prendre plusieurs boissons. Nous modélisons également un utilisateur qui paye avant de choisir sa boisson. Nous aurions pu également modéliser un utilisateur qui choisit sa boisson avant de payer.
 3. Les actions de la machine sont dans l'alphabet :

$$\{ \text{choix_boisson}, \text{annule}, \text{recup_jeton}, \text{servir_boisson} \} .$$

Le comportement de la machine peut être modélisé par l'automate représenté dans la figure A.4b (p. 304). Quelques remarques sur cet l'automate :

- Dans une modélisation plus réaliste, il faudrait modéliser les actions de l'utilisateur comme des entrées de cette machine. Il faudrait par ailleurs que cet automate soit complet sur ces entrées.
4. Cela est possible en faisant un produit entre l'automate de l'utilisateur et l'automate de la machine avec synchronisation sur les actions reliées. Par exemple, les actions caf, the, et choc de l'utilisateur doivent être synchronisées avec l'action *choix_boisson* de la machine.
 5. Les questions se traduisent comme suit.
 - Tous les comportements du système respectent la propriété : $L_{sys} \subseteq L_{prop}$.
 - Il est possible que le système réalise l'un des comportements de la propriétés : $L_{sys} \cap L_{prop} \neq \emptyset$.
 - Le système ne respecte pas la propriété : au choix, en fonction de la signification que l'on souhaite $L_{sys} \subseteq L_{prop}$ ou $L_{sys} \cap L_{prop} \neq \emptyset$.

6. Les algorithmes à utiliser sont comme suit.

- Pour vérifier $L_{sys} \subseteq L_{prop}$, on montre que $L_{sys} \cap \overline{L_{prop}} = \emptyset$. Pour cela, on complémente l'automate associé à L_{prop} et on montre que le langage associé au produit des automates est le langage vide;
- Pour vérifier $L_{sys} \cap L_{prop} \neq \emptyset$, on montre que le langage associé au produit des automates est non vide.
- C'est le même principe que pour les deux premiers cas en remplaçant L_{prop} par son complémentaire $\overline{L_{prop}}$.

7. D'abord, nous modélisons les propriétés comme suit :

- La propriété décrivant le fait que le client puisse obtenir une boisson sans avoir inséré la pièce est décrite par l'automate représenté dans la figure A.4c. La réponse à la question posée est positive s'il est possible que le système réalise un de ces comportements.
- Nous utilisons une propriété décrivant le fait que le client après avoir inséré un jeton et choisi une boisson paye à nouveau. Cette propriété est décrite par l'automate représenté dans la figure A.4d. La réponse à la question posée est positive s'il est possible que le système réalise un de ces comportements.
- Le système ne se bloque pas si tous les états de l'automate sont coaccessibles et le langage reconnu par le système est infini. Ainsi, à partir de n'importe quel état atteint, il existe toujours un prolongement de la séquence d'actions en cours qui mène à l'état initial et accepteur de l'automate où le système recommence une transaction.

Solution de l'exercice 171 (page 297)

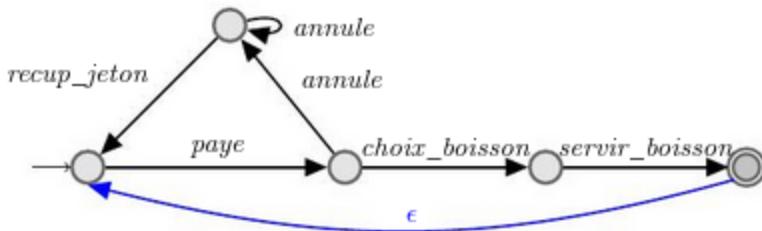
1. Nous détaillons l'ensemble des états Q qui représentent les configurations des protagonistes sur les deux côtés de la rivière. Nous avons quatre protagonistes B (le berger), L (le loup), C (la chèvre) et X (le chou) à placer sur un côté de la rivière. L'ensemble des protagonistes est $Protagonistes = \{B, L, C, X\}$ (abrégé $Prota$). Nous représentons les deux cotés de la rivière par un couple où le premier élément est le côté initial et le second élément l'autre côté. Ainsi, l'ensemble des configurations est un sous-ensemble de $\mathcal{P}(Protagonistes) \times \mathcal{P}(Protagonistes)$. La configuration initiale est : $(Protagonistes, \emptyset)$. Les configurations possibles sont celles où tous les participants sont présents, d'un côté ou l'autre de la rivière :

$$Possibles = \{(q_1, q_2) \in \mathcal{P}(Prota) \times \mathcal{P}(Prota) \mid q_1 \cup q_2 = Prota\}.$$

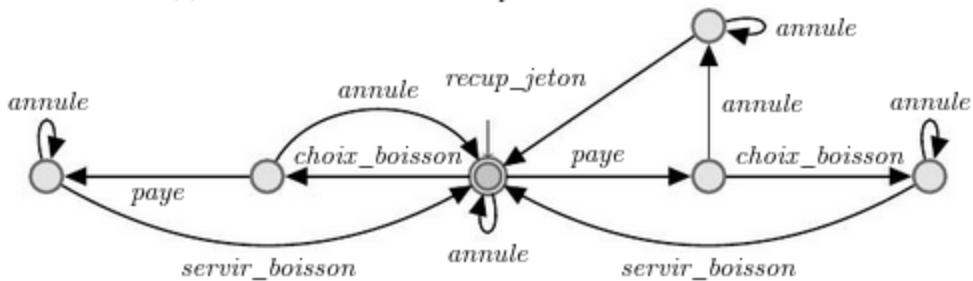
Certaines configurations sont impossibles, car un participant ne peut pas être des deux cotés de la rivière :

$$Impossibles = \{(q_1, q_2) \in \mathcal{P}(Protagonistes) \times \mathcal{P}(Protagonistes) \mid q_1 \cap q_2 \neq \emptyset\}.$$

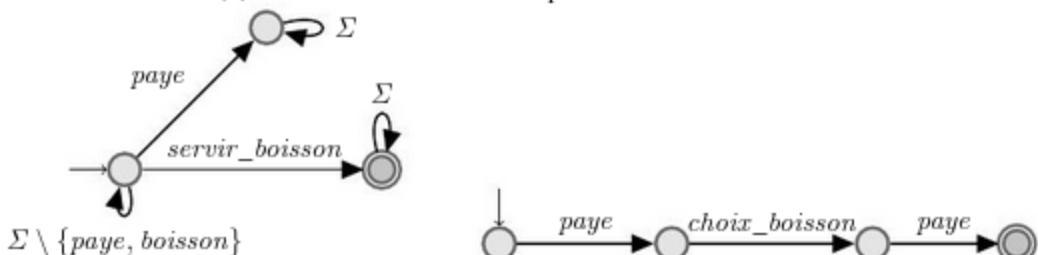
Certaines configurations sont des configurations d'erreur que l'on veut éviter (par exemple, la configuration où le loup mange la chèvre car ils sont du même côté de la



(a) Automate modélisant le comportement de l'utilisateur.



(b) Automate modélisant le comportement de la machine.



(c) Automate modélisant la propriété décrivant le fait que le client puisse obtenir une boisson sans avoir inséré de jeton.

(d) Automate modélisant la propriété décrivant le fait que client paye à nouveau après avoir inséré un jeton et choisi une boisson.

Figure A.4 – Automates solutions pour l'exercice 170.

rivière sans la surveillance du berger) :

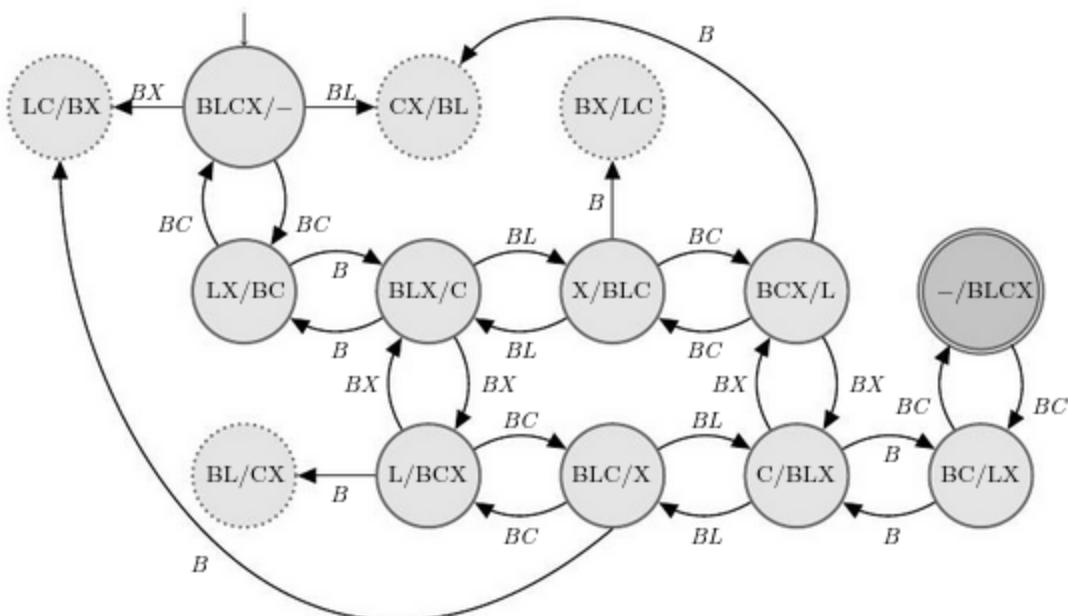
$$\begin{aligned}
 \text{Erreur} = \{(q_1, q_2) \in \mathcal{P}(\text{Prota}) \times \mathcal{P}(\text{Prota}) \mid & C \in q_1 \wedge L \in q_1 \wedge B \in q_2 \\
 & \vee C \in q_2 \wedge L \in q_2 \wedge B \in q_1 \\
 & \vee C \in q_1 \wedge X \in q_1 \wedge B \in q_2 \\
 & \vee C \in q_2 \wedge X \in q_2 \wedge B \in q_1\}.
 \end{aligned}$$

Au final, l'ensemble des configurations est : $\text{Possibles} \setminus \text{Impossibles} \setminus \text{Erreur}$. Nous noterons qu'en conséquence de la définition des configurations possibles, ce qui se passe sur du côté initial de la rivière détermine la situation globale : les participants absent du coté initial sont forcément de l'autre coté de la rivière.

2. Nous utilisons la représentation suivante : l'état représenté dans la figure A.5a représente la configuration où w et x sont sur la partie gauche de la rivière alors que y et z sont sur la partie droite. L'automate est représenté dans la figure A.5b. Les états en



(a) Représentation par un état d'une configuration des protagonistes : w et x sont d'un côté de la rivière et y et z de l'autre.



(b) Automate représentant l'évolution des protagonistes des deux côtés de la rivière en fonction des déplacements effectués par le berger.

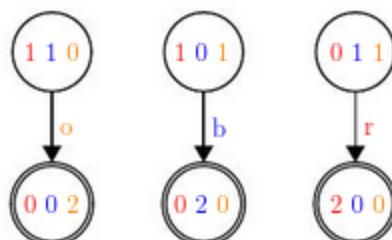
Figure A.5 – Solution de l'exercice 171.

pointillés sont des états d'erreur. L'automate n'est pas complet, en particulier, nous ne représentons pas de transitions sortant d'un état d'erreur. Le seul état accepteur est l'état $- / BLCX$ qui représente la configuration où tous les protagonistes ont traversé la rivière et sont tous du même côté.

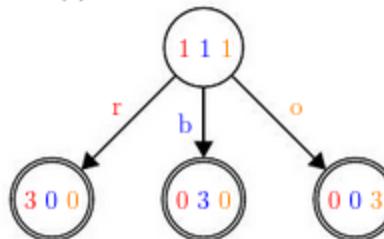
3. Nous pouvons utiliser l'automate trouvé à la question précédente pour résoudre le problème. Tout d'abord, nous déduisons l'existence d'une solution en utilisant l'algorithme pour décider du langage vide : il y a en effet un état accepteur et accessible dans l'automate. De plus, nous notons l'existence d'une infinité de solutions en utilisant l'algorithme pour décider du langage infini : il y a en effet un cycle accessible et coaccessible dans l'automate.

Solution de l'exercice 172 (page 297)

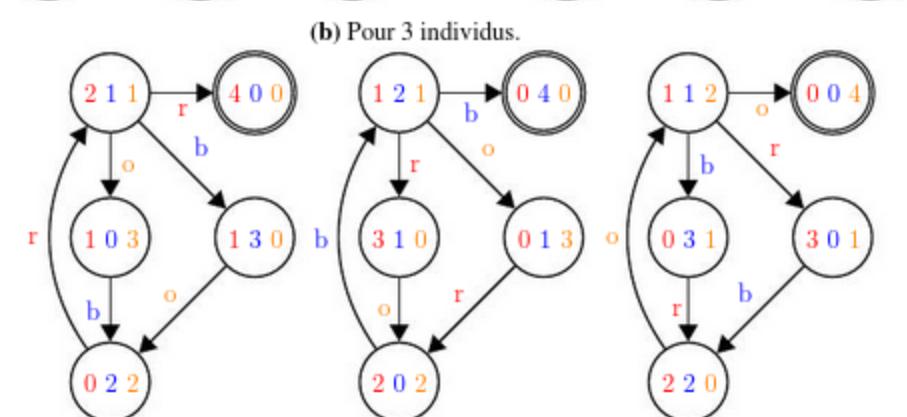
1. Ce sont des conséquences directes des règles.
2. Nous modélisons l'évolution de la population de la planète comme suit.



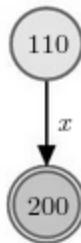
(a) Pour 2 individus.



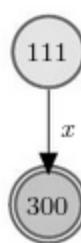
(b) Pour 3 individus.



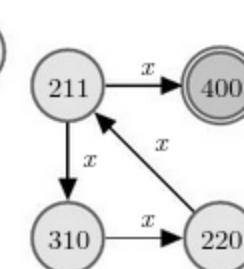
(c) Pour 4 individus.



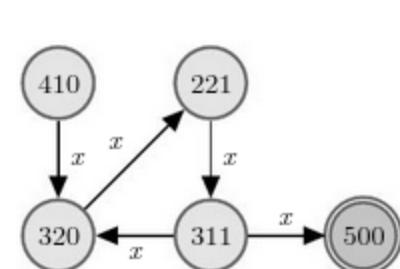
(d) Pour 2 individus.



(e) Pour 3 individus.



(f) Pour 4 individus.



(g) Pour 5 individus.

Figure A.6 – Automates solutions de l'exercice 172.

- Un état représente une configuration de la planète et contient 3 entiers qui sont les effectifs des espèces.
 - L'état initial représente les effectifs initiaux (équilibrés si possible).
 - Un état est accepteur si les espèces ne peuvent plus évoluer (c'est-à-dire une espèce a dominé les autres et il y a deux espèces avec effectif 0).
 - Une transition indique une naissance d'individus suite à une reproduction.
 - L'événement **rouge** (r) représente la naissance de deux individus **rouge** et la mort d'un individu **bleu** et d'un individu **orange**.
 - Les événements **bleu** (b) et **orange** (o) se définissent de manière analogue.
3. L'automate modélisant la planète avec 2 individus est représenté dans la figure A.6a. L'automate modélisant la planète avec 3 individus est représenté dans la figure A.6b. L'automate modélisant la planète avec 4 individus est représenté dans la figure A.6c.
 4. L'évolution de la planète ne dépend pas de l'ordre dans lequel apparaissent les trois entiers dans l'état. Plus précisément, les 6 configurations suivantes sont équivalentes du point de vue de l'évolution :

$$(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)$$

5. Pour que l'évolution s'arrête, il faut et il suffit que le nombre d'individus dans deux espèces soit égal à 0.
6. On utilise un multi-ensemble pour représenter une configuration en abstrayant l'information sur les effectifs des espèces. Par exemple, la configuration 210 signifie qu'il y a une espèce avec deux individus, une espèce avec un individu et une espèce avec aucun individu. Cette configuration abstrait les configurations (2, 1, 0), (1, 0, 2), (0, 1, 2), (1, 2, 0), (0, 2, 1) et (2, 0, 1). Pour de plus clarté, nous ordonnons les effectifs de la configuration abstraite par ordre décroissant.
7. Nous donnons l'évolution de la planète pour chaque nombre d'individus initiaux.
 - Pour 2 individus, l'automate est représenté dans la figure A.6d.
 - Pour 3 individus, l'automate est représenté dans la figure A.6e.
 - Pour 4 individus, l'automate est représenté dans la figure A.6f.
8. Pour 5 individus, l'automate est représenté dans la figure A.6g.
9.
 - a) L'évolution s'arrêtera inévitablement si tous chemins issus de l'état mène à un état accepteur (qui sont des états puits).
 - b) L'évolution ne peut pas s'arrêter si l'état courant est dans un cycle tel que aucun état du cycle ne soit coaccessible.
 - c) L'évolution peut s'arrêter si l'état courant est coaccessible

Solution de l'exercice 173 (page 298)

1. Nous définissons les actions suivantes :
 - C_A, C_B : pour compléter les récipients de 3 litres et 5 litres, respectivement,
 - V_A, V_B : pour vider les récipients de 3 litres et 5 litres, respectivement,

- T_{AB} (resp. T_{BA}) : pour transférer le contenu du récipient 1 vers le récipient 2 (resp. du récipient 2 vers le récipient 1).

Avec les actions ainsi définies, l'alphabet de l'automate est :

$$\{C_A, C_B, V_A, V_B, T_{AB}, T_{BA}\}.$$

2. Nous représentons un état par un couple (n_a, n_b) où n_a et n_b sont les nombres de litres contenus dans les récipients A et B , respectivement. L'espace d'états de l'automate est ainsi $[0, 3] \times [0, 5]$.
3. L'ensemble des états accepteurs est $[0, 3] \times \{4\}$, c'est-à-dire les états où le récipient 2 contient 4 litres quelque soit le nombre de litres contenus dans le récipient 1.
4. L'ensemble des transitions est défini par l'union des ensembles suivants :
 - $\{((n_a, n_b), C_A, (3, n_b)) \mid n_a \in [0, 3] \wedge n_b \in [0, 5]\}$ – pour compléter le récipient A de 3 litres ;
 - $\{((n_a, n_b), C_B, (n_a, 5)) \mid n_a \in [0, 3] \wedge n_b \in [0, 5]\}$ – pour compléter le récipient B de 5 litres ;
 - $\{((n_a, n_b), V_A, (0, n_b)) \mid n_a \in [0, 3] \wedge n_b \in [0, 5]\}$ – pour vider le récipient A de 3 litres ;
 - $\{((n_a, n_b), C_B, (n_a, 0)) \mid n_a \in [0, 3] \wedge n_b \in [0, 5]\}$ – pour vider le récipient B de 5 litres ;
 - $\{((n_a, n_b), T_{AB}, (n_a - c, n_b + c)) \mid c = \max\{c_1 \in \mathbb{N} \mid n_a - c_1 \geq 0 \wedge n_b + c_1 \leq 5\}\}$ – pour transférer depuis le récipient A vers le récipient B ;
 - $\{((n_a, n_b), T_{BA}, (n_a + c, n_b - c)) \mid c = \max\{c_1 \in \mathbb{N} \mid n_b - c_1 \geq 0 \wedge n_a + c_1 \leq 3\}\}$ – pour transférer depuis le récipient B vers le récipient A .
5. Il nous faut tester l'accessibilité d'un état accepteur, c'est-à-dire vérifier que le langage reconnu par l'automate est non vide. À partir de la fonction de transition donnée à la question précédente, nous pouvons adapter l'un des algorithmes de parcours (section 5.2.2), puis utiliser l'algorithme qui teste le langage vide (algorithme 11 (p. 100)). Si le langage reconnu par l'automate est non vide, alors il existe une solution au problème.
6. Par exemple, nous donnons ci-dessous la séquence des transitions menant à l'état accepteur $(3, 4)$:

$$(0, 0) \xrightarrow{C_A} (0, 5) \xrightarrow{T_{BA}} (3, 2) \xrightarrow{V_A} (0, 2) \xrightarrow{T_{BA}} (2, 0) \xrightarrow{C_B} (2, 5) \xrightarrow{T_{BA}} (3, 4).$$

Solution de l'exercice 174 (page 298)

1. Lorsque l'attaquant observe a , le système peut être dans l'état 1 ou 4. Puis, lorsque l'attaquant observe ensuite b , le système peut être dans l'état 2 ou 5.
2. En conséquence de l'existence de la séquence $a \cdot b$ qui révèle le secret, le système n'est pas opaque.

3. De manière similaire, nous examinons les séquences observables par l'attaquant qui peuvent être produites par le système. L'attaquant peut d'abord observer a et dans ce cas le système peut être dans l'état 2 ou 4. Puis, l'attaquant peut ensuite observer b et le système peut être dans l'état 3 ou 5. Puis, l'attaquant peut ensuite observer uniquement des a et si c'est le cas, le système est dans l'état 3.

Le système est donc opaque.

4. Nous définissons l'ensemble $E_w = \delta^*(q_{\text{init}}, w)$, c'est l'ensemble des états accessibles en lisant le mot w dans l'automate. Le mot w ne révèle pas le secret si $E_w \not\subseteq F$.
5. Le système est opaque si et seulement si $\forall w \in \Sigma^*. E_w \not\subseteq F$.
6. Oui pour cela, on construit l'automate $S_d = (Q_d, \Sigma, Q_{\text{init}}^D, \delta_d, F_d)$ qui est le déterminisé du système. Les états de ce système représentent la connaissance de l'attaquant suivant ses observations. Ainsi, le système est opaque si et seulement si $\forall q_d \in Q_d. q_d \not\subseteq F$ (rappel : les états du déterminisé sont des ensembles d'états de l'automate du système).

Bibliographie

- [1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.
- [2] Dean N. Arden. Delayed-logic and finite-state machines. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design*, pages 133–151. IEEE Computer Society, 1961.
- [3] Gérard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(3):117–126, 1986.
- [4] Janusz A. Brzozowski and Edward J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. Electronic Computers*, 12(2):67–76, 1963.
- [5] Olivier Carton. *Langages formels, calculabilité et complexité*. Vuibert, 2008.
- [6] Jérémie Dubreil, Philippe Darondeau, and Hervé Marchand. Supervisory control for opacity. *IEEE Trans. Automat. Contr.*, 55(5):1089–1100, 2010.
- [7] Samuel Eilenberg. *Automata, languages, and machines*. Pure and applied mathematics. Academic Press, 1976.
- [8] Yliès Falcone and Hervé Marchand. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems*, 25(4):531–570, 2015.
- [9] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.
- [10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.
- [11] John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley series in computer science and information processing. Addison-Wesley, 1969.
- [12] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.

- [13] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. *Automata studies. Annals of Mathematics Studies*, 34:3–41, 1956.
- [14] Jin Liu, Zhenhua Duan, Cong Tian, and Nan Zhang. An extended strange planet protocol. *J. Comb. Optim.*, 30(2):299–319, 2015.
- [15] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE Trans. Electronic Computers*, 9(1):39–47, 1960.
- [16] Dominique Perrin. Finite automata. In van Leeuwen [24], pages 1–57.
- [17] Dominique Perrin. Les débuts de la théorie des automates. *Séminaire de Philosophie et Mathématiques*, (1):1–17, 1993.
- [18] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
- [19] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
- [20] Jean-Pierre Ramis, André Warusfel, Xavier Buff, Josselin Garnier, Emmanuel Halberstadt, François Moulin, Monique Ramis, and Jacques Sauloy. *Mathématiques Tout-en-un pour la Licence 1*. Dunod, 2018.
- [21] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- [22] Loïc Teyssier, Jean-Romain Heu, and Gaël Collinet. *Mathématiques*. Dunod, 2017.
- [23] Cong Tian, Zhenhua Duan, and Jin Liu. Secure communications with strange planet protocol. *Optimization Letters*, 8(1):201–209, 2014.
- [24] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier and MIT Press, 1990.

Index

- absurde, *voir* preuve par l'absurde
acceptation d'un mot
 par un AFED, 48
 selon la fonction de transition étendue,
 48
 par un AFEND, 145
 par un ϵ -AFEND, 168
accessibilité, 46, *voir* état accessible
accessibilité dans les graphes finis, 99
accessible, *voir* état/automate accessible
AFED, *voir* automate déterministe
AFEND, *voir* automate non déterministe
algorithme, 98, 151
 d'élimination des états, 222
 de calcul de l'automate produit, 101
 de calcul des classes d'équivalence, 124
 de calcul des états accessibles, 95, 96
 de calcul des états coaccessibles, 96
 de complémentation, 101
 de complétion, 101
 de décision
 accessibilité, 99, 102
 coaccessibilité, 100, 102
 de la complétude, 101
 déterminisme, 151
 égalité de langages, 102
 finitude du langage, 100
 inclusion de langages, 102
 langage universel, 179
 langage vide, 100
de détection de cycle, 98
de déterminisation
 d'un ϵ -AFEND, 179
de distinction entre états, 122
de minimisation, 125
de parcours, 89, 151
 itératif, 93
 récuratif, 91, 92
terminaison, 91, 92, 94, 96, 97
alphabet, 29, 74, 101, 144, 168
antiréflexivité, *voir* relation antiréflexive
 de la relation de distinguabilité, 121
antisymétrie, *voir* relation antisymétrique
appartenance d'un élément à un ensemble, 22
application, 24, 46
Arden, *voir* lemme d'Arden
assertion logique, 19
 condition nécessaire, 21
 condition suffisante, 21
 conjonction, 20
 disjonction, 20
 équivalence, 20
 implication, 20
 négation, 20
 valeur de vérité, 19
associativité, 207
 d'un opérateur, 25
 de la concaténation, 32
 des opérateurs booléens, 21
 des opérateurs ensemblistes, 23
automate
 à nombre fini d'états, 29
 accessible, 97, 102, 122, 124
 coaccessible, 97, 102
 complémentaire, 74, 75, 101
 complémenté, 71, 72

- complet, 46, 49, 50, 52, 74, 101, 129
 complété, 72
 cycle, 273
 des dérivées, 228, 231
 correction, 228
 déterministe, 45, 46, 151, 169
 déterminisé, 145
 émondé, 97, 100, 102
 équivalent, 149
 minimal, 125, 127–129
 minimisé, 125
 non déterministe, 144, 168
 avec ϵ -transitions, 167, 168
 généralisé, 148
 normalisé, 222, 230
 produit, 71, 73, 75, 101
 quotient, 120, 125
- automate vers expression régulière, *voir* traduction
 automate vers grammaire, *voir* traduction
 axiome, 247
- bijection, *voir* fonction bijective
 Brzozowski, 220
- calcul
 des classes d'équivalence, 124
 des dérivées, 226, 231
 des états accessibles, 94
 des états coaccessibles, 95
 des langages associés aux chemins, 219, 223
 des langages associés aux états, 219, 220
- cardinal d'un ensemble, 25, 101
 Chomsky, *voir* classification de Chomsky
 classe d'équivalence, 23, 127, 128
 d'un entier, 26
 d'un état, 125
 classe des langages à états, 48, 167, 170
 classification de Chomsky, 247–249
 coaccessibilité, 46, *voir* état coaccessible
 coaccessibilité dans les graphes finis, 99
 coaccessible, *voir* état/automate coaccessible
 commutativité, 207
 des opérateurs booléens, 21
 des opérateurs ensemblistes, 23
- complémentaire ensembliste, 22
 complémentation, *voir* automate complémenté, 74, 75
 complétion, *voir* automate complété, 74
 complexité
 de la déterminisation, 146, 150, 151
 composition d' ϵ -AFEND, 177
 composition de grammaires, 251, 263
 complémentation, 261
 concaténation, 263
 fermeture de Kleene, 263
 intersection, 261
 union, 263
 concaténation, 29, 32, 205
 de langages, 31, 34
 de mots, 30
 élément neutre, 32
- condition
 nécessaire, 21, 33, 34, 103, 281
 nécessaire et suffisante, 33, 34
 suffisante, 21, 33, 34, 101–103
 de non-régularité d'un langage, 281, 283, 286
- configuration, 45, 50, 148
 acceptante, 47, 48, 50
 bloquante, 47, 50, 52
 d'un AFED, 47
 d'un AFEND, 144
 d'un ϵ -AFEND, 167, 168
 non acceptante, 47, 48, 50
 terminale, 47
- congruence entre entiers, 26
 conjonction d'assertions, 20
 constante d'itération, 273, 274, 282
 minimale, 273–275
- construction compositionnelle d'automate, 219, 225, 231
- contradiction, 274
- contraposée, *voir* preuve, 281
- convention
 gross-boutiste, 174, 251
 petit-boutiste, 174
- correction
 de l'élimination des ϵ -transitions, 169
 de la complémentation, 73
 de la complétion, 72

- de la déterminisation, 169
de la traduction automate vers grammaire, 260
de la traduction grammaire vers automate, 259
du produit, 73, 76
couple, 23
cycle, 89, 97, 98, 273
existence, 97
- décidabilité
de l'équivalence et de l'inclusion des langages, 220
de la finitude du langage, 100
du problème du langage vide, 100
- décidable, *voir* problème de décision décidable
- décomposition, 273, 274
- dérivation d'une grammaire
d'une grammaire
en une étape, 248
en plusieurs étapes, 248, 249
- dérivée d'une expression régulière
automate des dérivées, 228
ensemble des dérivées, 228
finitude, 228
par rapport à un mot, 228
par rapport à un symbole, 227
- détection de cycle, 89, 97
- déterminisation, 148, 149, 151, 167, 169, 175–177, 179
- déterminisé, 145, 169
- déterminisme, 45, 47
- Dirichlet, *voir* principe de Dirichlet
- disjonction d'assertions, 20
- distinguabilité, 126
entre AFED, 125
entre états, 119, 121
à k pas, 121
- distributivité, 207
- divisibilité, 26
- EF, *voir* classe des langages à états
- égalité de langages, 102, 129
- égalité ensembliste, 22
- élément absorbant
- de la conjonction, 21
de la disjonction, 21
des opérateurs ensemblistes, 23
- élément d'un ensemble, 22
- élément neutre
d'un groupe, 25
d'un monoïde, 25
de la conjonction, 21
de la disjonction, 21
des opérateurs ensemblistes, 23
- élément symétrique, 25
- élimination des ϵ -transitions, 167–169, 175–177
correction, 169
- élimination des états, 219, 221, 230
- ensemble, 22
cardinal, 25, 33, 34
complémentaire, 22
de mots, 30
- définition
en extension, 22
en intension, 22
inductive, 26
- dénombrable, 25
- des entiers naturels, 25
- des entiers relatifs, 25
- des parties d'un ensemble, 22
- élément, 22
appartenance à un ensemble, 22
non-appartenance à un ensemble, 22
- fini, 25
- infini, 25
- intersection, 22
- ordonné, 24
- quotient, 23
- union, 22
vide, 22, 25
- ensemble d'états, 144, 168, 169
- ensembles
distincts, 22
égaux, 22
en bijection, 25
équipotents, 25
- entier
congru, 26
divisible, 26

- naturel, 25
 relatif, 25
 représentation, 25
- ϵ -AFEND, *voir* automate non déterministe avec ϵ -transitions
- ϵ -fermeture, 167
 d'un ensemble d'états, 169
 d'un état, 169, 179
- ϵ -transition, 167
- équation, 231
 associée à un chemin, 230
 associée à un état, 220, 229
 résolution, 221
- équivalence
 à k pas, 123
 d'assertions, 20
 entre AFED, 119, 125, 176
 entre AFEND, 149, 150
 entre états, 119, 123
 entre expressions régulières, 207, 209
 entre grammaires et automates, 258
 opérateur, 20
- état, 45, 46, 50
 accepteur, 45, 46, 144, 168, 169
 accessible, 46, 48, 53, 93, 94, 102, 172
 coaccessible, 46, 48, 53, 95, 102
 distinguabilité, 121
 ϵ -fermeture, 169, 179
 équivalence, 123
 équivalent, 119
 initial, 45, 46, 144, 168, 169
 prédécesseur, 90, 95
 puits, 52, 72
 successeur, 90, 93, 94, 97
 terminal, *voir* état accepteur
- états
 distinguables, 126–129
 distingués, 119
 équivalents, 126–128
 jumeaux, 52
- exécution
 acceptée, 150
 d'un AFED, 47, 48, 50
 d'un AFEND, 145, 148, 150
 d'un ϵ -AFEND, 167, 168, 172
- existence d'un cycle, 97
- expression régulière
 associée à un chemin, 223
 équivalence, 207
 étendue, 228
 opérateur d'itération, 205
 opérateur de concaténation, 205
 opérateur d'union, 205
 sémantique, 205, 206, 232
 simplification, 207
 syntaxe, 205, 206
 vers automate, *voir* traduction
- extension, *voir* ensemble
 extension d'un mot, 30
 extension de la fonction de transition, 145
- facteur d'un mot, 30, 50, 274
 non vide, 274
- facteur itérant, 274, 282
- fermé (langage), *voir* fermeture
 fermeture
 de *EF*, *voir* fermeture des langages à états
 de Kleene, 31, 167, 171, 172, 207
 idempotence, 175
 propriétés, 175
 par extension, 31
 par facteur, 31, 174
 par opération miroir, 210
 par préfixe, 31
 par suffixe, 31, 174
 réflexive et transitive, 23
 transitive, 23
- fermeture des langages à états, 167, 170, 172, 174, 177
- par complémentation, 72, 73
 par concaténation, 167, 170
 par facteur, 174
 par fermeture de Kleene, 167, 172
 par image miroir, 167
 par intersection, 72, 73, 167
 par morphisme, 167, 171, 178
 par morphisme inverse, 167, 171, 178
 par opération miroir, 178
 par préfixe, 103
 par suffixe, 174
 par union, 72, 73, 167, 170

- fermeture des langages réguliers, 258, 281, 285
par complémentation, 258, 282, 284
par concaténation, 258
par fermeture de Kleene, 258
par intersection, 282, 284, 285
par morphisme, 282, 285
par morphisme inverse, 282
par union, 258, 282, 285
- file, 92
défilage, 92
enfilage, 92
- fonction, 24, 47
arité, 26
bijective, 24, 126
codomaine, 24
croissante, 25
domaine, 24
injective, 24
partielle, 24
point fixe, 24
surjective, 24
totale, 24
- fonction de renommage, 126
- fonction de transition, 45, 52
d'un AFED, 46
étendue, 172
étendue aux mots, 45, 48, 52
définition inductive, 48
extension, 145
partielle, 47
totale, 46
- forme sententielle, 248
- grammaire, 247
algébrique, 247
axiome, 247
composition, 251
contextuelle, 248
forme sententielle, 248
générale, 247, 248
langage généré, 248, 250, 251, 261
linéaire, 248
droite, 247, 248, 251, 257
gauche, 247, 248, 251, 257
mot généré, 248, 249, 251
- non contextuelle, 248
rationnelle, 248
règle de production, 247
régulière, 247, 248, 251, 257
sous contexte, 247
type, 248
vocabulaire, 248
non terminal, 247
terminal, 247
- grammaire vers automate, *voir* traduction
- grammaire vers expression régulière, *voir* traduction
- groupe, 25
commutatif, 25
- idempotence, *voir* opérateur idempotent, 207
- identités classiques entre expressions régulières, 209
- image d'un langage
par morphisme, 171, 178
par morphisme inverse, 171, 178
par opération miroir, 170, 178
- implication (opérateur), 20
- inclusion de langages, 102, 129
- inclusion ensembliste, 22
- indécidable, *voir* problème de décision indécidable
- induction, *voir* preuve par induction
- injection, *voir* fonction injective
- intension, *voir* ensemble
- intersection ensembliste, 22, 71
- intervalle, 25
- itération, 205, 273
constante, 274
lemme, 273
- Kleene, *voir* théorème de Kleene
- langage, 29, 30
à états, 48, 53, 76, 103
accepté, *voir* langage reconnu
algébrique, 248
cardinal, 99
complémentaire, 101
concaténation, 31
dénoté par une expression régulière, 206

- des mots d'une certaine longueur, 30
- fini, 99, 100
- généré par une grammaire, 248, 250, 251, 261
- hors contexte, 248
- infini, 99, 100
- miroir, 170, 178
- non régulier, 281, 283, 284
- préfixe-clos, 103
- rationnel, 29
- reconnu, 45, 50–53, 102, 129, 146, 148
 - par un AFED, 48, 49
 - par un AFEND, 145
 - par un ϵ -AFEND, 167, 168, 173
- régulier, 29, 206, 248, 273, 281
- universel, 30, 179
- vide, 34, 99
- lemme d'Arden, 219, 221
 - démonstration, 231
- lemme de l'itération, 273, 275, 281
 - utilisation, 281, 283, 284
- lettre, *voir* symbole
- loi de composition interne, 25
- longueur d'un mot, 30, 32
- McCluskey, 220
- méthode par élimination des états, 221
- minimalité, 125
 - du minimisé, 125
- minimisation, 119, 125, 127, 129, 177
- minimisé, 125, 127–129
 - correction, 125
- miroir
 - langage, 170, 178
 - mot, 170, 178
- monoïde, 25
- morphisme
 - appliqué à un langage, 171, 178
 - de groupe, 25
 - de mots, 171
 - inverse, 171
- mot, 29, 45
 - accepté, 45
 - par un AFED, 48
 - par un AFEND, 145, 146
 - par un ϵ -AFEND, 168, 172
- accepté/reconnu
 - par un ϵ -AFEND, 167
- concaténation, 30, 32
- définition applicative, 29
- définition inductive, 51
 - par la droite, 30
 - par la gauche, 30
- extension, 30
- facteur, 30, 274, 282
- généré, 248, 249, 251
- longueur, 30, 32
- miroir, 170, 178
- nombre d'occurrences d'un symbole, 30, 32
- non accepté, 48
 - par un AFED, 48
 - par un AFEND, 146
 - par un ϵ -AFEND, 172
- préfixe, 30
- suffixe, 30
- témoin de la distinguabilité, 122
- vide, 29, 30, 32
- négation (d'une assertion), 20
- nombre d'exécutions, 150
- nombre d'occurrences d'un symbole, 30, 32
- non-acceptation d'un mot
 - par un AFED, 48
 - selon la fonction de transition étendue, 48
 - par un AFEND, 145
- normalisation, 219, 222
 - pour l'initialisation, 222, 230
 - pour la terminaison, 222, 230
- opérateur
 - associatif, 21
 - booléen, 20
 - commutatif, 21
 - d'équivalence, 20
 - d'implication, 20
 - d'itération, 205
 - d'union, 205
 - de concaténation, 205
 - idempotent, 21
 - précédence, 207

- priorité, 207
opération miroir, 210
ou exclusif, 20
- parcours
 en largeur, 89, 92
 en profondeur, 89, 92, 97
 version itérative, 89
 version récursive, 89
 itératif, 90, 92
 récuratif, 90, 91
 stratégie, 92
- parenthèse, *voir* utilisation des parenthèses
- partie, 22
- partition, 23
- pas d'induction, 27
- pas de récurrence, 27
- pile, 92, 97
 dépilage, 93
 empilage, 93
- point fixe, 24
- prédecesseur, 89, *voir* état prédecesseur
- prédict, 21
 paramètre, 21
- préfixe d'un mot, 30, 50, 103
- preuve, 26
 par contraposition, 27
 par induction, 27
 cas de base, 27
 pas d'induction, 27
 par l'absurde, 27, 274, 281–283
 par le principe du tiers exclu, 26
 par récurrence, 27
 cas de base, 27
 faible, 27
 forte, 27
 pas de récurrence, 27
- principe de Dirichlet, *voir* principe des tiroirs
- principe des tiroirs, 24
- principe du tiers exclu, 26
- priorités des opérateurs, 21
- problème de décision, 90, 98
 de l'accessibilité, 99
 de l'équivalence entre expressions régulières, 207
 de la coaccessibilité, 99
- décidable, 99, 100, 103
 de la finitude du langage, 99
 du langage vide, 99
- équivalence des langages, 219
- inclusion de langages d'expressions régulières, 208
- inclusion des langages, 219
- indécidable, 99
- langage infini, 90
- langage vide, 90
- procédure de décision, 99, 100
- procédure de décision, 99, 100
- produit
 cartésien, 23
 d'automates, 73, 126
- produit d'automates, *voir* automate produit, 129
- propriété de fermeture
 utilisation, 282, 285
- quantification
 existentielle, 21
 universelle, 21
- quotient, *voir* automate quotient
- reconnaissance (de mot), 29
- récurrence, *voir* preuve par récurrence
- réflexivité, *voir* relation réflexive
 de la relation d'équivalence, 123
- règle de production, 247, 249
- relation, 23
 antiréflexive, 23
 antisymétrique, 23
 composition, 23
 d'équivalence, 23, 26, 123, 207
 d'ordre, 23, 25
 totale, 25
 usuelle entre entiers naturels, 25
- de distinguabilité, 121
- de transition, 144
- fermeture transitive, 248
- identité, 23
- réflexive, 23
- symétrique, 23
- transitive, 23
- relation de dérivation, 45, 47

- d'un AFED, 47, 50
d'un AFEND, 144
d'un ϵ -AFEND, 168
d'une grammaire, 248
 en plusieurs étapes, 248
 en une étape, 248
 en plusieurs étapes, 249
relation de transition, 168, 169
 d'un AFEND, 144, 151
 d'un ϵ -AFEND, 168
renommage des états, 102, 126, 224
représentation d'un automate
 graphique, 46
 tabulaire, 46, 49
- séquence, 24
séquence de configurations, *voir* exécution
séquence de symboles, *voir* mot
simplification d'expression régulière, 207, 210
solution d'un système d'équations, 221
sous-automate, 53
sous-ensemble, 22
successeur, 89, *voir* état successeur
suffixe d'un mot, 30, 50
suite, 24
sur-automate, 53
surjection, *voir* fonction surjective
symbole, 29, 45
 non terminal, 247
 puits, 52
 terminal, 247
symétrie, *voir* relation symétrique
 de la relation d'équivalence, 123
 de la relation de distinguabilité, 121
système d'équations associé à un automate,
 220
- table de vérité, 19, 20
taille d'une expression régulière, 210
technique de preuve, 26
terme constant, 227
 calcul, 227
terminaison, 91, 92, 94, 96, 97
théorème de Kleene, 219, 220
théorie des langages, 29
traduction
automate vers expression régulière, 220
automate vers grammaire, 260
expression régulière vers automate, 224
grammaire linéaire droite vers automate,
 259
grammaire linéaire gauche vers automate,
 259
grammaire vers automate, 259
grammaire vers expression régulière, 260
transformation d'automate, 231
transition, 45, 53, 97, 102
 arrière, 97
transitivité, *voir* relation transitive
 de la relation d'équivalence, 123
- union ensembliste, 22, 71, 205
universel, *voir* langage
utilisation des parenthèses, 207
- valeur de vérité, 19
vide, *voir* langage ou mot vide
 langage, 34
 mot, 30, 32
visite d'un état, 89–92, 97
vocabulaire, 247–249
 non terminal, 247
 terminal, 247