INFORMATIQUE THÉORIQUE
IFT-15751

Jules Desharnais
Université Laval
Québec

8 octobre 2004

Table des matières

Préface

La référence bibliographique la plus pertinente pour le cours est le livre de Brookshear [Brookshear89]. Certaines des preuves qui sont omises dans ces notes s'y trouvent. C'est du livre de Gries et Schneider [Gries93] que la méthode de présentation des preuves utilisée dans ces notes a été extraite.

La numérotation des exemples, exercices, définitions, ... est faite séquentiellement à l'intérieur de chaque chapitre. Ainsi, « exemple 1.13 » réfère à l'item 13 du chapitre 1, lequel item est un exemple.

Je sollicite vos suggestions pour l'amélioration de ces notes et j'aimerais que vous me souligniez les erreurs que vous y trouverez, les petites comme les grosses.

Je remercie

- Pierre Audet et François-Nicola Demers pour m'avoir aidé dans la rédaction de ce texte.
- Paul Goulet, de la faculté des Sciences de l'éducation, pour ses commentaires sur une ébauche de ces notes et pour son encouragement.
- L'université Laval et la faculté des Sciences et de génie pour le soutien financier accordé dans le cadre du *Programme institutionnel de soutien à l'innovation pédagogique* (1991–1992).

Jules DESHARNAIS

Depuis l'hiver 2000, je maintiens les notes de cours élaborées par M. DESHARNAIS. J'ai introduit des modifications mineures. J'ai modifié, plus particulièrement, l'expression du lemme de pompage et de son corollaire pour les langages non contextuels.

C'est à moi que vous pouvez vous adresser pour les suggestions d'amélioration et les erreurs.

Nadia TAWBI

Objectifs

Voici la liste des objectifs généraux du cours d'informatique théorique (énumération principale, numérique). Pour chaque objectif général, la sous-énumération alphabétique donne la liste des objectifs spécifiques qui lui sont rattachés. Les objectifs sont énoncés dans l'ordre de leur apparition dans ces notes. La notation [i] désigne un objectif intermédiaire menant à l'atteinte des autres objectifs.

1. Se rappeler les définitions et les concepts de base de la théorie des ensembles.
   (a) Dire si deux ensembles donnés sont égaux ou non et expliquer pourquoi. [i]
   (b) Distinguer les concepts d'appartenance, d'inclusion et d'inclusion stricte. [i]
   (c) Étant donné un ensemble A, construire l'ensemble puissance de A. [i]
   (d) Évaluer une expression constituée de plusieurs opérateurs ensemblistes. [i]
   (e) Énumérer les couples du produit cartésien de deux ensembles. [i]
   (f) Distinguer les concepts de relation et de fonction. [i]
   (g) Déterminer le domaine et le codomaine d'une relation. [i]
   (h) Étant données les trois propriétés des fonctions (injectivité, surjectivité, bijectivité), classifier des fonctions données selon les propriétés qu'elles possèdent. [i]
2. Appliquer les méthodes de preuve par contradiction et de preuve par induction.
   (a) Démontrer un énoncé donné au moyen d'une preuve par contradiction.
   (b) Démontrer un énoncé donné au moyen d'une preuve par induction.
3. Distinguer les notions d'ensembles finis, infinis, dénombrables et non dénombrables.
   (a) Étant donnés deux ensembles S et T, dire si la cardinalité de S est inférieure, égale ou supérieure à la cardinalité de T et expliquer pourquoi.
   (b) Dire si un ensemble donné est fini ou infini et expliquer pourquoi.

(c) Dire si un ensemble donné est dénombrable ou non et expliquer pourquoi.

4. Comprendre la notion de langage.

   (a) Dire si une séquence donnée appartient à un langage donné et expliquer pourquoi.

   (b) Décrire en français un langage exprimé formellement par une expression ensembliste.

   (c) Donner une expression ensembliste représentant un langage décrit par une phrase française.

5. Comprendre les capacités et les limites des automates finis en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires régulières et les expressions régulières.

   (a) Démontrer le fonctionnement d'un automate fini déterministe donné en énumérant les configurations du ruban et les états consécutifs à une configuration initiale donnée. [i]

   (b) Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate fini déterministe donné. [i]

   (c) Décrire le langage accepté par un automate fini déterministe donné.

   (d) Construire un automate fini déterministe acceptant un langage donné.

   (e) Démontrer qu'un langage donné n'est pas régulier.

   (f) Classifier des automates finis décrits par des diagrammes de transitions. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix. [i]

   (g) Classifier des descriptions formelles d'automates finis. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix. [i]

   (h) Démontrer le fonctionnement d'un automate fini donné en énumérant les configurations du ruban et les états consécutifs à une configuration initiale donnée. [i]

   (i) Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate fini donné. [i]

   (j) Transformer un automate non déterministe en automate déterministe.

   (k) Décrire le langage accepté par un automate fini donné.

   (l) Construire un automate fini acceptant un langage donné.

   (m) Décrire le langage généré par une grammaire régulière donnée.

   (n) Construire une grammaire régulière générant un langage donné.

(o) Décrire le langage représenté par une expression régulière donnée.

(p) Construire une expression régulière représentant un langage donné.

(q) Les grammaires régulières, les expressions régulières et les automates finis décrivent les mêmes langages : Passer d'une représentation à l'autre.

6. Comprendre les capacités et les limites automates à pile en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires non contextuelles.

   (a) Démontrer le fonctionnement d'un automate à pile donné en énumérant les configurations du ruban et celles de la pile ainsi que les états consécutifs à une configuration initiale donnée. [i]

   (b) Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate à pile donné. [i]

   (c) Transformer un automate qui accepte sans vider sa pile en automate qui vide sa pile avant d'accepter. [i]

   (d) Décrire le langage accepté par un automate à pile donné.

   (e) Construire un automate à pile acceptant un langage donné.

   (f) Décrire le langage généré par une grammaire non contextuelle donnée.

   (g) Construire une grammaire non contextuelle générant un langage donné.

   (h) Passer d'une grammaire non contextuelle à un automate à pile.

   (i) Convertir une grammaire non contextuelle quelconque en grammaire sous forme normale de Chomsky.

   (j) Donner l'arbre de dérivation d'une séquence donnée par une grammaire non contextuelle donnée.

   (k) Classifier des automates à pile décrits par des diagrammes de transitions. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix. [i]

   (l) Classifier des descriptions formelles d'automates à pile. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix. [i]

   (m) Construire un automate à pile déterministe acceptant un langage donné.

   (n) Démontrer qu'un langage donné n'est pas un langage non contextuel.

   (o) Construire et utiliser des tables d'analyse $LL(1)$.

   (p) Utiliser des tables d'analyse $LR(1)$.

7. Comprendre les capacités et les limites des machines de Turing (et donc des ordinateurs) en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires à structure de phrase.

(a) Démontrer le fonctionnement d'une machine de Turing donnée en énumérant les configurations du ruban et les états consécutifs à une configuration initiale donnée. [i]

(b) Démontrer le fonctionnement d'une machine de Turing donnée sous la forme d'un schéma de combinaison de machines de Turing en énumérant les configurations du ruban consécutives à une configuration initiale donnée. [i]

(c) Classifier des machines de Turing décrites par des diagrammes de transitions. Dire tout d'abord si la description est correcte. Si oui, dire si la machine est déterministe. Expliquer la raison de chaque choix. [i]

(d) Classifier des descriptions formelles de machines de Turing. Dire tout d'abord si la description est correcte. Si oui, dire si la machine est déterministe. Expliquer la raison de chaque choix. [i]

(e) Décrire le langage accepté par une machine de Turing donnée.

(f) Construire une machine de Turing acceptant un langage donné.

(g) Transformer une machine de Turing donnée en une machine équivalente utilisant des conventions différentes pour l'acceptation.

(h) Expliquer comment transformer une machine de Turing en une machine équivalente utilisant des conventions différentes pour l'acceptation.

(i) Décrire le langage généré par une grammaire à structures de phrase donnée.

(j) Construire une grammaire à structures de phrase générant un langage donné.

(k) Construire une machine de Turing décidant un langage donné.

(l) Déterminer si un langage donné est décidable (récursif).

(m) Déterminer si un langage donné est un langage à structures de phrase (est récursivement énumérable).

Chapitre 0

Préliminaires et révision

OBJECTIFS GÉNÉRAUX

1. Se rappeler les définitions et les concepts de base de la théorie des ensembles.
2. Appliquer les méthodes de preuve
   - par contradiction,
   - par induction.
3. Distinguer les notions d'ensembles finis, infinis, dénombrables et non dénombrables.

Au coeur du cours d'informatique théorique se trouve la notion de langage. Intuitivement, un langage n'est rien de plus qu'un ensemble de phrases, de chaînes, de mots qui sont eux-mêmes constitués d'éléments d'un autre ensemble : l'alphabet. Puisque les objets de notre étude, les langages, sont définis à partir des ensembles, nous allons consacrer à ces derniers une courte révision.

0.1 Concepts de base de la théorie des ensembles

OBJECTIFS SPÉCIFIQUES

1. Dire si deux ensembles donnés sont égaux ou non et expliquer pourquoi.
2. Distinguer les concepts d'appartenance, d'inclusion et d'inclusion stricte.
3. Étant donné un ensemble A, construire l'ensemble puissance de A.
4. Évaluer une expression constituée de plusieurs opérateurs ensemblistes.
5. Énumérer les couples du produit cartésien de deux ensembles.
6. Distinguer les concepts de relation et de fonction.
7. Déterminer le domaine et le codomaine d'une relation.
8. Étant données les trois propriétés des fonctions (injectivité, surjectivité, bijectivité), classifier des fonctions données selon les propriétés qu'elles possèdent.

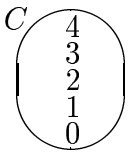
Le texte est constitué principalement de définitions, d'exemples illustrant les définitions et de simples exercices d'application. Faites les exercices au fur et à mesure que vous les rencontrez.

0.1.1 Définitions et notation

0.1 Définition. Un *ensemble* est une collection (famille) d'objets. □

Les objets d'un ensemble sont aussi désignés sous le nom d'éléments ou de membres. En général les éléments d'un ensemble ont en commun une ou plusieurs propriétés qui les caractérisent. On utilise les accolades ($\{,\}$) pour décrire un ensemble. Par exemple, on écrit $S = \{a, b, c\}$ pour indiquer que l'ensemble S est constitué des éléments a, b, c. La description d'un ensemble (ce qu'il contient) s'effectue le plus couramment par la définition des propriétés de ses objets, par l'énumération de ses éléments ou par un diagramme.

0.2 Exemple. Soient $A = \{n : n \in \mathbf{N} \text{ et } n < 5\}$, $B = \{0, 1, 2, 3, 4\}$ et



L'ensemble A est défini en donnant les propriétés de ses éléments ; l'ensemble B est défini par l'énumération de ses éléments ; finalement, C est défini à partir d'un diagramme. Notez que $A = B = C$. On peut informellement décrire un ensemble infini en donnant une liste de ses premiers éléments, suivie de ..., lorsque le contexte permet de comprendre ce que sont les éléments non énumérés. Par exemple, l'ensemble des entiers positifs pairs est $\{2, 4, 6, 8, \ldots\}$. □

Les éléments d'un ensemble peuvent eux-même être des ensembles, comme par exemple l'ensemble $\{\{a\}, \{b\}, \{a, b\}\}$ dont les éléments sont $\{a\}$, $\{b\}$ et $\{a, b\}$. Finalement les ensembles peuvent contenir à la fois des ensembles et des non-ensembles, comme $\{a, 1, \{a\}, \{b\}\}$ qui a comme éléments a, 1, $\{a\}$ et $\{b\}$.

À titre de rappel, le tableau suivant présente les principaux ensembles de nombres :

Notation	Nom	Description
$\mathbf{N}$	Les naturels	Entiers positifs ou nuls
$\mathbf{N}^+$	Les naturels positifs	Entiers positifs
$\mathbf{Z}$	Les relatifs	Entiers positifs, négatifs ou nuls
$\mathbf{Q}$	Les rationnels	Nombres exprimables par un rapport d'entiers
$\mathbf{R}$	Les réels	Limites de suites de rationnels
$\mathbf{Q}'$	Les irrationnels	Nombres réels non exprimables par un rapport d'entiers

0.3 Définition. L'*ensemble vide*, noté $\{\}$ ou $\emptyset$, est l'ensemble qui ne contient aucun élément. □

0.4 Exercice. Est-ce que les ensembles suivants sont vides ? Justifiez votre réponse.

1. $\{\emptyset\}$
2. $\{x \in \mathbf{Z} : x < 0 \text{ et } x^2 < 0\}$
3. $\{\ \{\}\ \}$

□

0.5 Définition. Deux ensembles sont égaux ssi[1] ils ont les mêmes éléments. □

[1] Abréviation de « si et seulement si ».

0.6 Exemple. Les ensembles $\{a, b, c\}$ et $\{c, a, b\}$ sont égaux puisqu'ils ont les mêmes éléments, c'est-à-dire a, b et c. Les ensembles sont donc des collections inordonnées d'éléments. □

0.7 Exemple. Les ensembles $\{1, 1, 2, 2, 2, 2, \ 5, 5, 5\}$ et $\{1, 1, 1, 2, 5, 5\}$ sont égaux parce qu'ils ont les mêmes éléments, c'est-à-dire 1,2 et 5. Les ensembles sont donc indépendants du nombre d'occurrences de chacun de leurs éléments. □

0.8 Exercice. Les égalités suivantes sont-elles vraies ou fausses ? Dites pourquoi.

1. $\{\{1\}, \{2\}\} = \{1, 2\}$
2. $\{\{a\}, \{a, b\}\} = \{\{a\}, \{b, a\}, \{b, a\}\}$
3. $\emptyset = \{\emptyset\}$
4. $\{\emptyset\} = \{\{\}\}$
5. $\{1, 3, 5, 7, \ldots\} = \{n \in \mathbf{N} : (\exists m \in \mathbf{N} : n = 2m + 1)\}$

□

0.9 Définition. L'*appartenance* d'un objet e à un ensemble E est notée $e \in E$. On dit aussi que e est un *élément* (ou un *membre*) de E. La notation $e \notin E$ signifie que e n'est pas élément de (n'appartient pas à, n'est pas membre de) l'ensemble E. □

0.10 Définition. La relation $\subseteq$ s'appelle *inclusion*. On dit qu'un ensemble A est *inclus* dans un ensemble B, ce qui est noté $A \subseteq B$, si tous les éléments de A sont des éléments de B. On dit aussi que A est un *sous-ensemble* de B. La notation $A \not\subseteq B$ signifie que A n'est pas un sous-ensemble de B. □

L'inclusion peut être utile par exemple pour prouver que deux ensembles sont égaux. Il suffit dans ce cas de montrer que chacun des deux ensembles est inclus dans l'autre. En effet, deux ensembles A et B sont égaux ssi $A \subseteq B$ et $B \subseteq A$.

Les notions d'appartenance et d'inclusion ensemblistes sont souvent confondues. Les exemples et exercices qui suivent devraient permettre de bien les différencier.

0.11 Exemple. Soient $A = \{a, b\}$, $B = \{a, b, c\}$ et $C = \{\emptyset, \{a\}, \{b\}\}$. On peut voir que

- $a \in A$, $a \in B$, $c \notin A$, $c \in B$, $A \subseteq B$, $A \not\subseteq C$ et $B \not\subseteq A$;
- $\emptyset \in C$, puisque les éléments de C sont $\emptyset$, $\{a\}$ et $\{b\}$;
- $\emptyset \subseteq A$, $\emptyset \subseteq B$ et $\emptyset \subseteq C$ ($\emptyset \subseteq E$ pour n'importe quel ensemble E).

□

0.12 Exercice. Dites si les expressions suivantes sont vraies ou fausses.

1. $\{1\} \in \{1, 2\}$
2. $1 \in \{1, 2\}$
3. $\{1\} \subseteq \{1, 2\}$
4. $1 \subseteq \{1, 2\}$

□

L'inclusion stricte, notée $\subset$, est un cas particulier de l'inclusion ($\subseteq$). L'énoncé $A \subseteq B$ signifie que l'ensemble A est inclus ou est égal à l'ensemble B tandis que $A \subset B$ indique que A est inclus dans B mais que A et B ne sont pas égaux. On dit alors que A est un sous-ensemble propre (ou strict) de B.

Il est parfois utile de considérer des combinaisons d'éléments d'un ensemble. Soit un ensemble A. La collection de toutes les combinaisons possibles des éléments de A forme ce qu'on appelle l'ensemble puissance de A.

0.13 Définition. Soit A, un ensemble. L'*ensemble puissance* de A, noté $\wp(A)$, est l'ensemble de tous les sous-ensembles de A. □

0.14 Exemple. L'ensemble puissance de $\{0, 2, 4\}$ est

$$\{\emptyset, \{0\}, \{2\}, \{4\}, \{0, 2\}, \{2, 4\}, \{0, 4\}, \{0, 2, 4\}\}$$

□

0.15 Exercice. Calculez les ensembles puissances suivants :

1. $\wp(\{a, b, c, d\})$
2. $\wp(\emptyset)$
3. $\wp(\{\{a\}, \emptyset\})$

□

0.1.2 Opérations sur les ensembles

0.16 Définition. L'*union* des ensembles A et B est l'ensemble des éléments qui sont dans A ou B ou les deux. On note $A \cup B = \{x : x \in A \text{ ou } x \in B\}$. □

0.17 Exemple. $\{1, 2, 3\} \cup \{3, 4, 5\} = \{1, 2, 3, 4, 5\}$. □

0.18 Définition. L'*intersection* des ensembles A et B est l'ensemble des éléments contenus à la fois dans A et B. On note $A \cap B = \{x : x \in A \text{ et } x \in B\}$. □

0.19 Exemple. $\{1, 2, 3\} \cap \{3, 4, 5\} = \{3\}$. □

0.20 Définition. La *différence* entre les ensembles A et B, notée $A - B$, est l'ensemble qui contient les éléments qui sont dans A mais pas dans B. Autrement dit, $A - B = \{x : x \in A \text{ et } x \notin B\}$. □

0.21 Exemple. $\{1, 2, 3\} - \{3, 4, 5\} = \{1, 2\}$. □

L'ensemble $A - B$ est appelé le *complément de B relativement à A*. Le complément d'un ensemble est toujours relatif à un autre ensemble. Si A est sous-entendu, on dit simplement que $A - B$ est le complément de B. Par exemple, dans une discussion au sujet des nombres naturels, le complément de $\{2, 3, 4\}$ est $\{0, 1, 5, 6, 7, \ldots\}$.

0.22 Exercice. Évaluez les expressions suivantes.

1. $((\{a, b, c\} \cup \{c, d, e, f\}) \cap \{b, c, d, g, h\}) - \{c\}$
2. $\{\emptyset, \{a\}, \{b\}\} - \{a, \{b\}\}$
3. $(\{i \in \mathbf{N} : i > 5\} \cap \{j \in \mathbf{N} : j < 15\})$
   $- (\{j \in \mathbf{N} : j \text{ est pair}\} \cup \{k \in \mathbf{N} : k \text{ est divisible par } 3\})$
4. $\{\emptyset\} - \emptyset$

□

0.1.3 Relations et fonctions

Les concepts de relation et de fonction sont intimement liés aux ensembles. En effet ces concepts ne sont que des règles qui assignent aux éléments d'un ensemble des éléments d'un autre ensemble (ou du même). On rencontre de telles règles tous les jours ; par exemple, la règle qui assigne à chacun des étudiants d'une classe $\{\text{Georges, Joseph, Julie}, \ldots\}$ un élément de l'ensemble des cotes $\{\text{A}^+, \text{A}, \text{A}^-, \text{B}^+, \ldots\}$ est une fonction de l'ensemble des noms vers l'ensemble des cotes.

0.23 Définition. Soient les ensembles A et B. Le *produit cartésien* de A et B, noté $A \times B$, est l'ensemble de toutes les paires ordonnées (a, b) où $a \in A$ et $b \in B$. □

0.24 Exemple. $\{a, b\} \times \{1, 2, 3\} = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$
$\{1, 2\}^2 = \{(1, 1), (1, 2)(2, 1)(2, 2)\}$ □

0.25 Exercice. Calculez $\{1, 2, 3, 4\} \times \{i, j, k\}$.

□

La notion de produit cartésien peut être généralisée à plusieurs ensembles :

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) : a_i \in A_i \text{ pour } i = 1, 2, \ldots, n\}.$$

On écrit A^n au lieu de $\underbrace{A \times A \times \ldots \times A}_{n \text{ fois}}$.

0.26 Exemple. $\{x, y\} \times \{k\} \times \{a, b\} = \{(x, k, a), (x, k, b), (y, k, a), (y, k, b)\}$ □

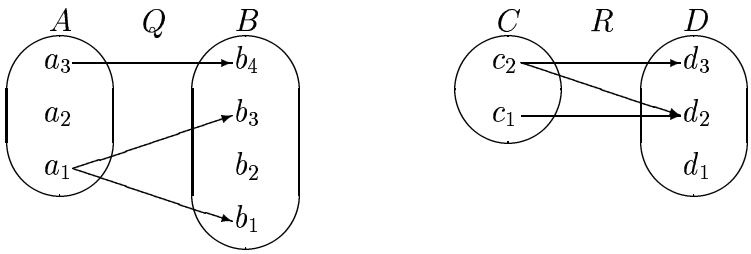
L'égalité $A \times \emptyset = \emptyset$ est un cas particulier du produit cartésien. Supposons par exemple que $A = \{a, b\}$. Selon la définition du produit cartésien, $A \times \emptyset$ est l'ensemble des paires telles que le premier élément est a ou b et dont le deuxième élément est dans $\emptyset$. Puisque l'ensemble vide ne contient aucun élément, il n'existe aucune paire, donc $A \times \emptyset = \emptyset$.

0.27 Définition. Une *relation* R entre les ensembles A et B est un sous-ensemble du produit cartésien $A \times B$. C'est-à-dire, $R \subseteq A \times B$. □

Une relation est caractérisée par

1. un ensemble de départ (A),
2. un ensemble d'arrivée (B),
3. un ensemble de couples vérifiant la relation (le sous-ensemble du produit cartésien).

0.28 Exemple. Voici deux relations :



Les ensembles de départ et d'arrivée ainsi que les couples de la relation Q précédente sont respectivement A, B et $(a_3, b_4), (a_1, b_1), (a_1, b_3)$. Les ensembles départ et d'arrivée de R sont C et D. Les couples de R sont $(c_2, d_3), (c_2, d_2), (c_1, d_2)$. On a donc

$$Q = \{(a_3, b_4), (a_1, b_1), (a_1, b_3)\} \subseteq A \times B,$$
$$R = \{(c_2, d_3), (c_2, d_2), (c_1, d_2)\} \subseteq C \times D.$$

□

Deux autres ensembles sont obtenus à partir des ensembles de départ, d'arrivée et des couples de la relation. Ce sont le domaine et le codomaine.

0.29 Définition. Le *domaine* d'une relation R, noté $\mathsf{dom}(R)$, est l'ensemble des éléments de l'ensemble de départ qui apparaissent comme première composante d'au moins une paire de la relation. Formellement,

$$\mathsf{dom}(R) = \{e : (\exists e' : (e, e') \in R)\}.$$

Le *codomaine* d'une relation R, noté $\mathsf{codom}(R)$, est l'ensemble des éléments de l'ensemble d'arrivée qui apparaissent comme deuxième composante d'au moins une paire de la relation. Formellement,

$$\mathsf{codom}(R) = \{e' : (\exists e : (e, e') \in R)\}.$$

□

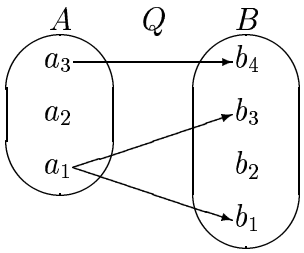
0.30 Exemple. Soient $S = \{2, 4, 6\}$, $T = \{0, 2, 8, 9, 11\}$ et $R = \{(x, y) \in S \times T : x \text{ est un diviseur exact de } y\}$.

- L'ensemble de départ est S.
- L'ensemble d'arrivée est T.
- La relation R est $\{(2, 0), (4, 0), (6, 0), (2, 2), (2, 8), (4, 8)\}$.
- $\mathsf{dom}(R) = \{2, 4, 6\}$.
- $\mathsf{codom}(R) = \{0, 2, 8\}$.

□

Une *image* d'un élément a par une relation R est un élément b tel que $(a, b) \in R$. L'*ensemble image* d'un élément a est l'ensemble de ses images. L'ensemble image d'un ensemble est l'union des ensembles images de ses éléments. Le codomaine d'une relation est donc l'image de son domaine.

0.31 Exemple. Soit Q, la relation suivante :



- b_1 et b_3 sont des images de a_1.
- $\{b_1, b_3\}$ est l'ensemble image de a_1.
- $\{b_4\}$ est l'ensemble image de a_3.

- Le codomaine, $\{b_1, b_3, b_4\}$, est l'image du domaine, $\{a_1, a_3\}$.

□

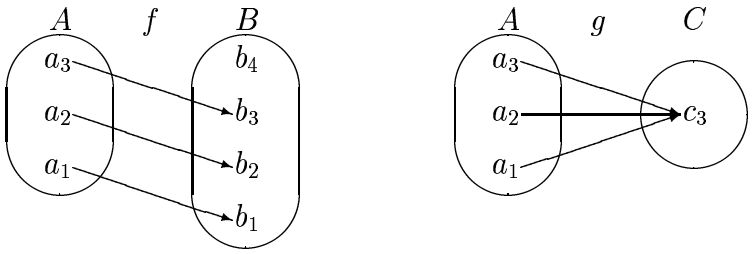
0.32 Exercice. Quelle est l'ensemble image de l'élément a_2 dans l'exemple 0.31 ? □

0.33 Définition. Une *fonction* f d'un ensemble A dans un ensemble B est une relation ayant A comme ensemble de départ et B comme ensemble d'arrivée, et telle que chaque élément de A apparaît *une et une seule fois* comme première composante d'une paire de f. Autrement dit, f est une fonction ssi les deux conditions suivantes sont satisfaites :

- $(\forall a \in A : (\exists b \in B : (a, b) \in f))$,
- $(\forall a \in A, b \in B, b' \in B : (a, b) \in f \wedge (a, b') \in f \Rightarrow b = b')$.

□

0.34 Exemple. Les relations f et g ci-dessous sont des fonctions.



La relation Q de l'exemple 0.31 n'est pas une fonction, et ce pour deux raisons :

- L'élément a_2 n'est associé à aucun élément de l'ensemble d'arrivée. Autrement dit, il n'y a aucune paire de Q qui a a_2 comme premier élément.
- L'élément a_1 est associé à deux éléments de l'ensemble d'arrivée. Autrement dit, il y a deux paires de Q avec a_1 comme premier élément ; ces deux paires sont (a_1, b_1) et (a_1, b_3).

□

Les fonctions ne sont que des relations plus restrictives. Une relation peut avoir plusieurs couples avec le même premier élément (par exemple $(2, 0), (2, 2), (2, 8)$) tandis que la définition de fonction l'interdit. Comme [Brookshear89], nous exigeons aussi qu'une fonction soit totalement définie, c'est-à-dire qu'elle soit définie pour chacun des éléments de son ensemble de départ. Cependant, d'autres auteurs n'imposent pas cette condition. Ces auteurs désignent plutôt les fonctions totalement définies par les noms de *fonctions totales* ou *applications*. Les fonctions qui ne sont pas totales sont appelées *fonctions partielles*.

0.35 Exercice. Donnez l'ensemble de départ, l'ensemble d'arrivée, le domaine et le codomaine pour les relations de l'exemple 0.28 et pour les fonctions de l'exemple 0.34.

	départ	arrivée	domaine	codomaine
Q				
R				
f				
g				

□

0.36 Notation. La notation $f : A \to B$ signifie que f est une fonction de A dans B. On peut écrire $b = f(a)$ au lieu de $(a, b) \in f$. □

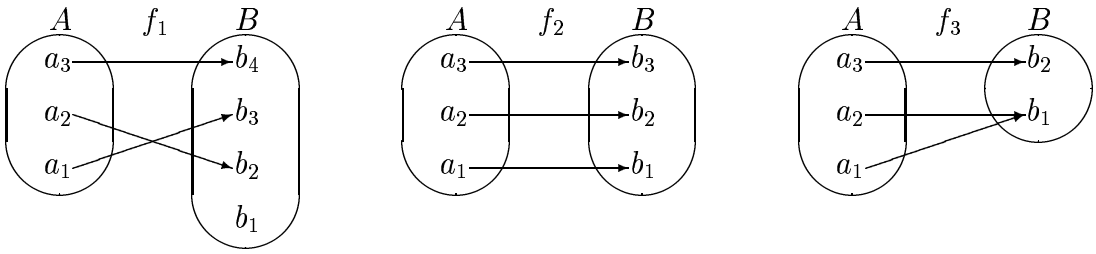
0.37 Exercice. Soit $f : A \to B$. Donnez, si possible,

1. l'ensemble de départ de f :
2. l'ensemble d'arrivée de f :
3. le domaine de f :
4. le codomaine de f :

□

0.38 Définition. Une fonction est *injective* ssi $x \neq y$ implique $f(x) \neq f(y)$, c'est-à-dire ssi deux éléments distincts de l'ensemble de départ ont deux images différentes. □

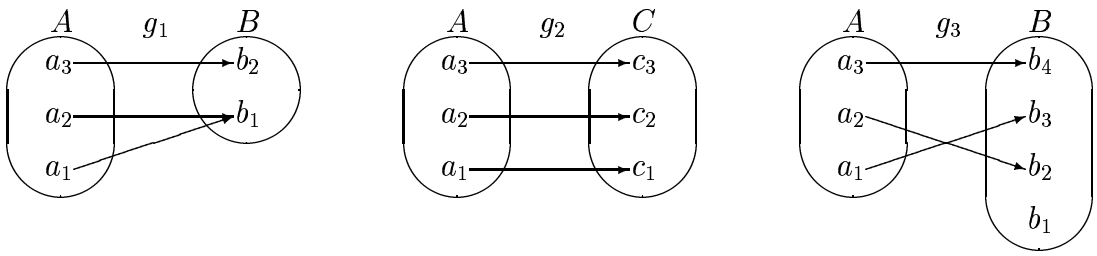
0.39 Exemple. Les fonctions f_1 et f_2 suivantes sont injectives parce que chaque élément de leur ensemble d'arrivée est l'image d'un ou d'aucun élément de leur ensemble de départ. La fonction f_3 n'est pas injective, car l'élément b_1 est l'image des éléments a_1 et a_2.



□

0.40 Définition. Une fonction est *surjective* ssi chaque élément de l'ensemble d'arrivée est l'image d'au-moins un élément du domaine. □

0.41 Exemple. Les fonctions g_1 et g_2 suivantes sont surjectives puisque pour chacune d'elle, chaque élément de l'ensemble d'arrivée est l'image d'au-moins un élément du domaine. Par contre, g_3 n'est pas surjective, puisque l'élément b_1 n'est l'image d'aucun élément du domaine.



□

0.42 Définition. Une fonction est *bijective* ssi elle est injective et surjective c'est-à-dire ssi chaque élément de l'ensemble d'arrivée est l'image d'un et d'un seul élément du domaine. □

0.43 Exercice. Parmi les fonctions des exemples 0.39 et 0.41, dites quelles sont les fonctions bijectives.

□

0.44 Exercice. Pour chacun des trois types de fonction (injective, surjective, bijective), dites si le codomaine est toujours égal à l'ensemble d'arrivée.

1. fonctions injectives
2. fonctions surjectives
3. fonctions bijectives

□

0.1.4 Problèmes

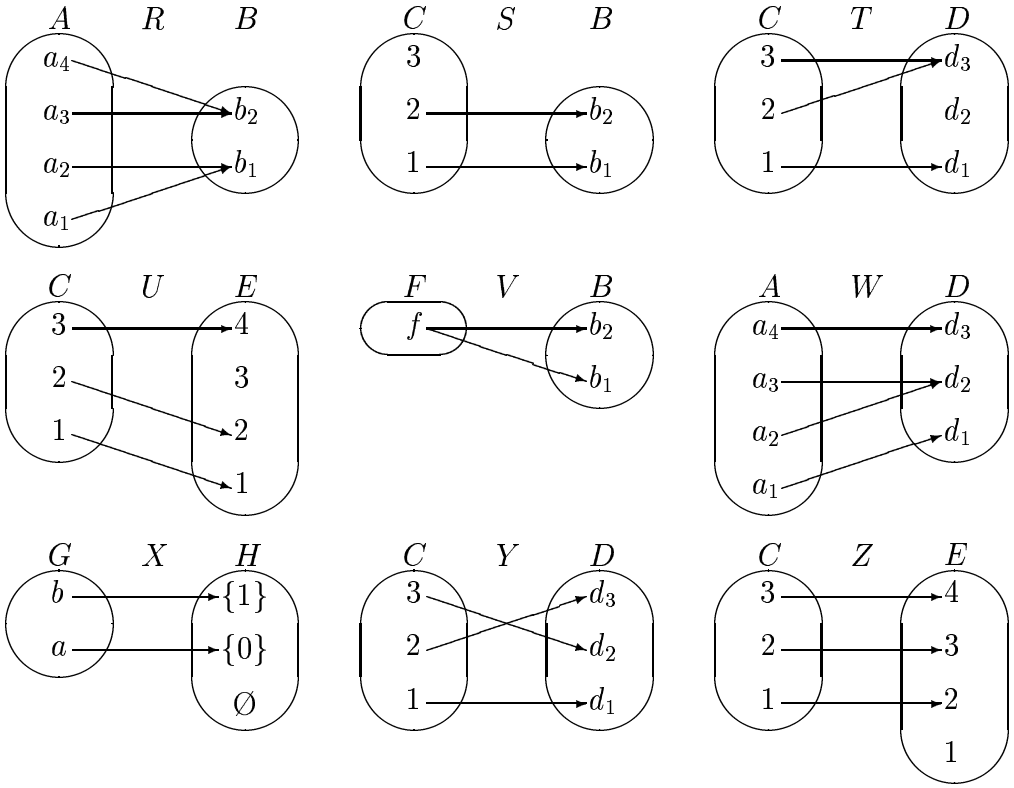
1. Vrai ou faux ?
   (a) Un ensemble peut contenir des éléments qui sont non corrélés. Par exemple $\{a, 1, \text{Fred}, \text{New Jersey}\}$.
   (b) Si $A \subseteq B$ et $B - A \neq \emptyset$ alors $A \subset B$.
   (c) $A = B$ si $\wp(A) = \wp(B)$.
   (d) Le domaine d'une relation est toujours égal à son ensemble de départ.
2. Dites si les expressions suivantes sont vraies ou fausses. Expliquez.
   (a) $\{\{1, 2\}\} = \{1, 2\}$

(b) $\{1,2\} \subseteq \{\{1,2\}\}$
(c) $\{1,2\} \in \{\{1,2\}\}$
(d) $\{1,2\} \in \mathbf{N}$
(e) $\emptyset \notin \{1,2\}$
(f) $\{1,2\} \subseteq \{a,b,1,2,3\}$
(g) $\{1,2\} \not\subseteq \{1,a,b\}$
(h) $\{n \in \mathbf{N} : \exists m \in \mathbf{N} : n = m^2\} = \{n \in \mathbf{N} : \exists m \in \mathbf{N} : m = n^2\}$
(i) $\{n \in \mathbf{N} : \exists m \in \mathbf{N} : n = 1 + m \text{ modulo } 2\} \subseteq \{1,2\}$

3. Dites si chacun des ensembles suivants est un ensemble puissance d'un ensemble. Si oui, quel est-il ?
   (a) $\emptyset$
   (b) $\{\emptyset, \{a\}\}$
   (c) $\{\emptyset, \{a\}, \{\emptyset, a\}\}$
   (d) $\{\emptyset, \{a\}, \{b\}, \{a,b\}\}$
4. Soient $A, B, C \subseteq U$. Complétez le tableau qui suit résumant les propriétés des opérations sur les ensembles :

Propriété	Nom
$A \cup \emptyset =$ $A \cap U =$	Élément neutre
$A \cup U =$ $A \cap \emptyset =$	Élément absorbant
$A \cup A =$ $A \cap A =$	Idempotence
$\overline{(\overline{A})} =$	Involution
$A \cup B =$ $A \cap B =$	Commutativité
$A \cup (B \cup C) =$ $A \cap (B \cap C) =$	Associativité
$A \cap (B \cup C) =$ $A \cup (B \cap C) =$	Distributivité
$\overline{A \cup B} =$ $\overline{A \cap B} =$	Lois de De Morgan

5. Soit $A = \{1, 2, 3\}$. Calculez $A \cap \wp(A)$.
6. Quel est $\mathbf{N} \cap \wp(\mathbf{N})$?
7. Calculez les expressions suivantes.
   - $\{\emptyset, a\} \times \{1, 2\} \times \{i, j\}$
   - $\{a, b\}^3$
8. Dites si chacune des relations suivantes est une fonction. Dans l'affirmative, dites aussi les propriétés que la fonction possède (injectivité, surjectivité, bijectivité).



0.2 Méthodes de preuve

Une preuve est la démonstration qu'un énoncé est vrai. Sa fonction première est d'éliminer toute réserve concernant la validité d'une proposition, d'un théorème, etc. Une preuve aide aussi le lecteur à comprendre pourquoi un résultat est vrai en le reliant aux hypothèses de départ et aux autres éléments de la théorie.

Dans le cadre du cours, deux méthodes de preuve seront fréquemment utilisées : preuve par contradiction et preuve par induction. Nous allons les étudier en détail afin de comprendre leur formulation et leur logique. Ainsi, les démonstrations des théorèmes à venir seront plus faciles à comprendre.

OBJECTIFS SPÉCIFIQUES

1. Démontrer un énoncé donné au moyen d'une preuve par contradiction.
2. Démontrer un énoncé donné au moyen d'une preuve par induction.

Rappelons d'abord les opérateurs logiques et leur table de vérité. Ces opérateurs sont les suivants :

Symbole	Opération	Prononciation	Priorité
$\neg$	négation	$\neg A$: non A	4
$\wedge$	conjonction	$A \wedge B$: A et B	3
$\vee$	disjonction	$A \vee B$: A ou B	2
$\Rightarrow$	implication	$A \Rightarrow B$: A implique B	1
$\Leftarrow$	conséquence	$A \Leftarrow B$: A est une conséquence de B	1
$\Leftrightarrow$	équivalence	$A \Leftrightarrow B$: A est équivalent à B	1

Nous assignons un ordre de priorité aux opérateurs logiques, afin de diminuer le nombre des parenthèses nécessaires; les opérateurs ayant la priorité la plus élevée sont évalués en premier. Cette priorité est indiquée dans la table ci-dessus.

Voici maintenant la table de vérité des opérateurs logiques.

0.45 Table de vérité.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftarrow B$	$A \Leftrightarrow B$
vrai	vrai	faux	vrai	vrai	vrai	vrai	vrai
vrai	faux	faux	faux	vrai	faux	vrai	faux
faux	vrai	vrai	faux	vrai	vrai	faux	faux
faux	faux	vrai	faux	faux	vrai	vrai	vrai

0.46 Quantificateurs Nous ferons aussi un usage restreint des quantificateurs existentiel ($\exists$) et universel ($\forall$). Il est bon de se souvenir des deux lois suivantes, qui sont des généralisations des lois de De Morgan :

$$\neg(\forall x : p(x)) \Leftrightarrow (\exists x : \neg p(x)),$$
$$\neg(\exists x : p(x)) \Leftrightarrow (\forall x : \neg p(x)).$$

La portée des quantificateurs s'étend aussi loin que le permettent les parenthèses.

Lorsque ce sera possible, nous disposerons les preuves de la manière suivante :

$$\begin{array}{ll} & \text{expression}_0 \\ \text{op}_1 & \langle\ \text{justification}_1\ \rangle \\ & \text{expression}_1 \\ \text{op}_2 & \langle\ \text{justification}_2\ \rangle \\ & \text{expression}_2 \\ \ldots & \langle\ \ldots\ \rangle \\ \text{op}_n & \langle\ \text{justification}_n\ \rangle \\ & \text{expression}_n, \end{array}$$

où $\text{op}_i, i = 1 \ldots n$, est un opérateur relationnel transitif $(=, <, >, \leq, \geq, \subseteq, \subset, \ldots)$ ou un opérateur logique binaire, et où la justification_i explique comment on peut passer de l'expression_{i-1} à l'expression_i.

0.2.1 Preuve par contradiction

Prouver par contradiction qu'un énoncé P est vrai se résume à montrer $\neg P \Rightarrow$ faux. Si l'on regarde la table de vérité suivante, on comprend pourquoi la démonstration de $\neg P \Rightarrow$ faux suffit à établir la vérité de P ; en effet, la table indique que ces deux propositions sont équivalentes.

P	$\neg P$	$\neg P \Rightarrow$ faux	$P \Leftrightarrow (\neg P \Rightarrow$ faux$)$
faux	vrai	faux	vrai
vrai	faux	vrai	vrai

0.47 Exemple. Montrons que $\sqrt{2}$ est irrationnel, c'est-à-dire qu'on ne peut pas l'exprimer par un rapport d'entiers. Appelons P la proposition « $\sqrt{2}$ est irrationnel » et montrons que $\neg P \Rightarrow$ faux.

$\neg P$

$\Leftrightarrow$ $\langle$ La négation de P est « $\sqrt{2}$ est rationnel ». Notons que nous utilisons ici $\Leftrightarrow$, parce qu'on a bien l'équivalence, mais qu'il suffirait d'utiliser $\Rightarrow$. $\rangle$

$\sqrt{2}$ rationnel

$\Leftrightarrow$ $\langle$ Un nombre rationnel positif peut s'exprimer comme le rapport de deux nombres naturels positifs sans facteurs communs (1 n'est pas considéré comme un facteur commun). $\rangle$

$\exists a, b \in \mathbf{N}^+ : \sqrt{2} = a/b \wedge a$ et b n'ont pas de facteur commun

$\Leftrightarrow$ $\langle$ En élevant les deux côtés de l'équation au carré. $\rangle$

$\exists a, b \in \mathbf{N}^+ : 2 = a^2/b^2 \wedge a$ et b n'ont pas de facteur commun

$\Leftrightarrow$ $\langle$ En multipliant les deux côtés de l'équation par b^2. $\rangle$

$\exists a, b \in \mathbf{N}^+ : 2b^2 = a^2 \wedge a$ et b n'ont pas de facteur commun

$\Rightarrow$ $\langle$ Cela signifie que a^2 est pair, d'où a est pair aussi. Puisque a est pair, $a = 2c$ pour un certain entier c. $\rangle$

$\exists a, b, c \in \mathbf{N}^+ : 2b^2 = a^2 \wedge a = 2c \wedge a$ et b n'ont pas de facteur commun

$\Rightarrow$ $\langle$ Simple substitution. $\rangle$

$\exists a, b, c \in \mathbf{N}^+ : 2b^2 = 4c^2 \wedge a = 2c \wedge a$ et b n'ont pas de facteur commun

$\Leftrightarrow$ $\langle$ En divisant les deux côtés de la première équation par 2. $\rangle$

$\exists a, b, c \in \mathbf{N}^+ : b^2 = 2c^2 \wedge a = 2c \wedge a$ et b n'ont pas de facteur commun

$\Rightarrow$ $\langle$ Cela signifie que b^2 est pair et donc que b aussi est pair. $\rangle$

$\exists a, b, c \in \mathbf{N}^+ : b^2 = 2c^2 \wedge a = 2c \wedge b$ est pair

$\wedge\ a$ et b n'ont pas de facteur commun

$\Leftrightarrow$ $\langle$ Nous avons la contradiction annoncée. $a = 2c$ signifie que a est pair. Puis que b aussi est pair, a et b ont un facteur commun (2). Nous avons donc « a et b ont un facteur commun » et « a et b n'ont pas de facteur commun ». Cette proposition a la forme $p \wedge \neg p$. Elle donc fausse et elle entraîne la fausseté de toute la proposition quantifiée (voir la table de vérité 0.45). $\rangle$

$\exists a, b, c \in \mathbf{N}^+ :$ faux

$\Leftrightarrow$ $\langle$ La notation $\exists s \in S : p(s)$ signifie qu'il existe un s dans S tel que $p(s)$ est vrai. Or il ne peut y avoir de s tel que faux soit vrai. $\rangle$

faux.

□

Parfois les propositions à prouver auront la forme $A \Rightarrow B$. Par exemple, la proposition

Si un entier naturel x est pair et plus grand que 2 alors x est composé

a cette forme :

$x \in \mathbf{N} \wedge x$ est pair $\wedge\, x > 2 \Rightarrow x$ est composé.

0.48 Exercice. Transformez en implication la proposition suivante : Si n est un entier positif tel que la somme de ses diviseurs est $n + 1$, alors n est un nombre premier.

□

Pour démontrer un énoncé de la forme $A \Rightarrow B$ par contradiction, il faut montrer (tout comme ci-dessus) que $\neg(A \Rightarrow B) \Rightarrow$ faux.

0.49 Exercice. Montrez que $\neg(A \Rightarrow B) \Leftrightarrow (A \wedge \neg B)$.

□

0.50 Exemple. Démontrons par contradiction que si n est un entier positif tel que la somme de ses diviseurs est $n + 1$, alors n est un nombre premier.

Pour quelques nombres premiers, on voit bien que cette proposition est raisonnable :

n	Diviseurs(n)	$\sum$ Diviseurs(n)	n premier ?	($\sum$ Diviseurs(n)) $= n+1$?
2	$1, 2$	3	Oui	Oui
3	$1, 3$	4	Oui	Oui
4	$1, 2, 4$	7	Non	Non
5	$1, 5$	6	Oui	Oui
6	$1, 2, 3, 6$	12	Non	Non
7	$1, 7$	8	Oui	Oui
8	$1, 2, 4, 8$	15	Non	Non
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Allons-y pour la preuve.

$\neg$ (si n est un entier positif tel que la somme de ses diviseurs est $n+1$, alors n est un nombre premier)

$\Leftrightarrow$ $\langle$ Simple reformulation sous forme d'implication. $\rangle$

$\neg(n \in \mathbf{N}^+ \wedge (\sum \text{Diviseurs}(n)) = n+1 \Rightarrow n$ est un nombre premier)

$\Leftrightarrow$ $\langle$ Selon l'exercice 0.49 $\rangle$

$n \in \mathbf{N}^+ \wedge (\sum \text{Diviseurs}(n)) = n+1 \wedge \neg$ (n est un nombre premier)

$\Rightarrow$ $\langle$ Si n n'est pas un nombre premier, alors n possède au moins un diviseur d différent de 1 et n. La somme des diviseurs de n est donc supérieure ou égale à $n+d+1 > n+1$. $\rangle$

$n \in \mathbf{N}^+ \wedge (\sum \text{Diviseurs}(n)) = n+1 \wedge (\sum \text{Diviseurs}(n)) > n+1$

$\Leftrightarrow$ $\langle$ Quel que soit k, $k = n+1 \wedge k > n+1$ est faux. $\rangle$

faux.

□

0.51 Exercice. Montrez par contradiction que pour n'importe quels entiers naturels i, j et n, si $ij = n$ alors $i \leq \sqrt{n}$ ou $j \leq \sqrt{n}$.

□

0.2.2 Preuve par induction

Il existe beaucoup d'équations impliquant une variable qui appartient à l'ensemble des nombres naturels. Par exemple, pour tout $n \in \mathbf{N}, 2^{n+1} - 1 = \sum_{i=0}^{n} 2^i$.

Pour la plupart d'entre nous, essayer une telle formule avec les premières valeurs de n suffit pour se convaincre de sa validité. Par exemple :

$$\begin{array}{ll} n = 0 & 2^1 - 1 = 2^0 \\ n = 1 & 2^2 - 1 = 2^0 + 2^1 \\ n = 2 & 2^3 - 1 = 2^0 + 2^1 + 2^2 \\ n = 3 & 2^4 - 1 = 2^0 + 2^1 + 2^2 + 2^3 \\ n = 4 & 2^5 - 1 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 \end{array}$$

Ces quelques essais ne constituent cependant pas une preuve parce que rien nous ne assure que l'équation est valide pour tout n.

La méthode de preuve par induction permet de démontrer la validité d'équations du même type que la précédente. Soit $P(n)$ un énoncé quelconque impliquant une variable entière n. Pour prouver par induction que $P(n)$ est vrai pour tout $n \geq n_0$ il faut montrer les deux propositions suivantes :

1. $P(n_0)$ est vrai,
2. pour tout $k \geq n_0$, $P(k) \Rightarrow P(k+1)$.

Ces deux éléments d'une preuve par induction sont respectivement désignés sous les noms de *base d'induction* et d'*étape d'induction*. La base d'induction sert à montrer que l'énoncé est vrai pour la plus petite valeur de n considérée. Ensuite, à l'étape d'induction, on suppose que $P(k)$ est vrai (on appelle $P(k)$ l'*hypothèse d'induction*) et on montre que cela entraîne que $P(k+1)$ est vrai, quel que soit k. Selon le principe d'induction, cela suffit pour montrer que $P(n)$ est vrai pour tout $n \in \mathbf{N}$.

Cela est très raisonnable. En effet, considérons le cas usuel $n_0 = 0$ et supposons que nous ayons montré

1. $P(0)$ et
2. pour tout k, $P(k) \Rightarrow P(k+1)$.

Nous avons directement $P(0)$ (par 1). Mais comme $P(0) \Rightarrow P(1)$ (par 2), nous avons aussi $P(1)$. Comme $P(1) \Rightarrow P(2)$ (par 2), nous avons aussi $P(2)$, etc.

En termes plus formels, le principe d'induction assume tout simplement la validité de la formule suivante (autrement dit, la formule est un axiome) :

$$P(n_0) \wedge (\forall k \geq n_0 : P(k) \Rightarrow P(k+1)) \Rightarrow (\forall n \geq n_0 : P(n)).$$

0.52 Exemple. Montrons par induction $(\forall n \in \mathbf{N} : 2^{n+1} - 1 = \sum_{i=0}^{n} 2^i)$. Soit $P(n)$ le prédicat défini par

$$P(n) \Leftrightarrow 2^{n+1} - 1 = \sum_{i=0}^{n} 2^i.$$

Il faut montrer qu'il est vrai pour tout $n \in \mathbf{N}$.

- Base d'induction. Il s'agit de montrer que $P(n)$ est vrai pour la plus petite valeur de n soit $n = 0$. Donc, montrons que $P(0)$ est vrai, c'est-à-dire que $2^{0+1} - 1 = \sum_{i=0}^{0} 2^i$:

$$2^{0+1} - 1 = 1 = 2^0 = \sum_{i=0}^{0} 2^i.$$

- Étape d'induction. Montrons $P(k) \Rightarrow P(k+1)$. (Comme nous n'imposons pas de restriction à k, la preuve est bonne pour tout k.)

$$\begin{aligned}
& P(k) \\
\Leftrightarrow \quad & \langle \text{ Par définition de } P. \rangle \\
& 2^{k+1} - 1 = \textstyle\sum_{i=0}^{k} 2^i \\
\Rightarrow \quad & \langle \text{ Ajout de } 2^{k+1} \text{ de chaque côté. } \rangle \\
& 2^{k+1} - 1 + 2^{k+1} = (\textstyle\sum_{i=0}^{k} 2^i) + 2^{k+1} \\
\Leftrightarrow \quad & \langle \text{ Réorganisation des termes. } \rangle \\
& 2^{k+2} - 1 = \textstyle\sum_{i=0}^{k+1} 2^i \\
\Leftrightarrow \quad & \langle \text{ Réorganisation des termes. } \rangle \\
& 2^{(k+1)+1} - 1 = \textstyle\sum_{i=0}^{k+1} 2^i \\
\Leftrightarrow \quad & \langle \text{ Par définition de } P. \rangle \\
& P(k+1)
\end{aligned}$$

□

0.53 Exercice. Montrez par induction $(\forall n \in \mathbf{N} : n < 2^n)$.

□

0.2.3 Problèmes

1. Transformez en implications les propositions suivantes.
   (a) Si l'atmosphère était composée de 100% d'oxygène alors elle serait irrespirable pour beaucoup d'organismes.
   (b) Si $A \cap B = \emptyset$ et si $C \subseteq B$ alors $A \cap C = \emptyset$.
2. Soient A, B et C des ensembles quelconques. Montrez par contradiction que si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$.
3. Montrez par induction que si $x \neq y$, alors $x^n - y^n$ est divisible par $x - y$, pour tout n dans $\mathbf{N}^+$.
4. Montrez par induction que si x est un nombre réel non négatif, alors $(1+x)^n - 1 \geq nx$ pour tout n dans $\mathbf{N}$.
5. Montrez par induction $n! > 2^n$, pour tout $n \geq 4$.

0.3 Notion d'infini

OBJECTIFS SPÉCIFIQUES

1. Étant donnés deux ensembles S et T, dire si la cardinalité de S est inférieure, égale ou supérieure à la cardinalité de T et expliquer pourquoi.
2. Dire si un ensemble donné est fini ou infini et expliquer pourquoi.
3. Dire si un ensemble donné est dénombrable ou non et expliquer pourquoi.

0.3.1 Cardinalité d'un ensemble

Intuitivement, la cardinalité d'un ensemble est la « taille » de cet ensemble, ou la « quantité d'éléments » qu'il contient. La cardinalité d'un ensemble S est dénotée $|S|$. Nous allons préciser cette notion, afin de pouvoir l'appliquer autant aux ensembles finis qu'aux ensembles infinis.

0.54 Notation. Soit $n \in \mathbf{N}$. Définissons l'ensemble C_n de la manière suivante :

$$C_n = \{i \in \mathbf{N} : 0 \leq i < n\}.$$

□

0.55 Exercice. Décrivez les ensembles suivants en énumérant leurs éléments :

1. $C_5 =$
2. $C_1 =$
3. $C_0 =$

□

0.56 Définition. Pour tout $n \in \mathbf{N}$, on a $|C_n| = n$. □

De cette définition, on voit que la cardinalité d'un ensemble C_n est tout simplement le nombre d'éléments qu'il contient. Pour pourvoir parler de la cardinalité des autres ensembles, nous allons d'abord expliquer comment comparer les cardinalités.

0.57 Exercice. Montrez par induction $(\forall n \in \mathbf{N} : |\wp(C_n)| = 2^n)$.

□

Si E est un ensemble quelconque, il est évident que $|\wp(E)|$ dépend seulement de la cardinalité de E et pas des éléments particuliers de E. Il découle donc de l'exercice précédent que si E est un ensemble fini (notion que nous préciserons sous peu), alors $|\wp(E)| = 2^{|E|}$.

0.58 Définition. On dira que la cardinalité d'un ensemble S est *inférieure ou égale à* la cardinalité d'un ensemble T (dénoté par $|S| \leq |T|$) ssi il existe une fonction *injective* $f : S \to T$.

On dira que la cardinalité d'un ensemble S est *égale à* la cardinalité d'un ensemble T (dénoté par $|S| = |T|$) ssi il existe une fonction *bijective* $f : S \to T$.

On dira que la cardinalité d'un ensemble S est *inférieure à* la cardinalité d'un ensemble T (dénoté par $|S| < |T|$) ssi $|S| \leq |T|$ et $|S| \neq |T|$. □

Autrement dit, on a $|S| \leq |T|$ si et seulement si *chaque* élément de S peut être associé à un *unique* élément de T. Si, en plus de cela, *chaque* élément de T est associé à un élément de S, on a $|S| = |T|$.

0.59 Exemple. Soient les ensembles $A = \{4, 8, 3\}$, $B = \{c, d, a, e\}$, $C = \{a, b, c\}$.

1. On a $|A| \leq |B|$, car la fonction $f : A \to B$ suivante est injective :

   $$f = \{(4, c), (8, d), (3, a)\}.$$

   Il n'est pas possible de trouver une fonction surjective de A dans B. Supposons que l'on veuille ajouter une paire à f pour atteindre l'élément $e \in B$. Pour maintenir l'injectivité, il faudrait associer e à un élément de A qui ne soit pas déjà associé à un élément de B. Mais il n'y en a aucun. Remarquons qu'on peut aussi écrire $|B| \geq |A|$.

2. On a $|A| = |C|$, car la fonction $g : A \to C$ suivante est bijective :

   $$g = \{(4, c), (8, a), (3, b)\}.$$

3. On a $|\mathbf{N}| = |\mathbf{N}^+|$. En effet, la fonction $s : \mathbf{N} \to \mathbf{N}^+$ définie par $s(m) = m + 1$ (ou, de manière équivalente, par $s = \{(m, n) : n = m + 1\}$) est bijective.

□

0.60 Exercice.

1. Quelle est la cardinalité de l'ensemble suivant ? Justifiez votre réponse.

   $$\{\{a\}, \{b, c\}, \{d, e, f\}\}$$

   Le but de cet exercice simple est de voir que, dans le cas des ensembles finis (notion dont la définition se rapproche !), la définition 0.58 mène à une valeur de la cardinalité qui est celle à laquelle on s'attend. Ceci aide à accepter les résultats que nous obtenons dans le cas des ensembles infinis.

2. Comparez les cardinalités des ensembles suivants et justifiez votre réponse.

   $S = \{k \in \mathbf{N} : (\exists j \in \mathbf{N} : k = 2j)\}$
   $\mathbf{N}$

□

Il est facile de voir que la cardinalité du produit cartésien d'ensembles finis égale le produit des cardinalités de chacun des ensembles du produit cartésien. En d'autres mots,

$$|A \times B| = |A| \times |B|.$$

0.61 Exercice. Quelle est la cardinalité de $\{\{0, 1\}, \{a, b\}\} \times \{\alpha, \beta, \gamma\}$?

□

0.3.2 Ensembles finis et infinis

Nous avons une idée intuitive de ce qu'est un ensemble fini ou infini. Par exemple, nous savons que l'ensemble $\{4, 5, 6, 7\}$ est fini, alors que $\mathbf{N}$ et $\mathbf{R}$ sont infinis. Nous allons maintenant donner une définition rigoureuse de cette notion.

0.62 Définition. Un ensemble S est dit *fini* ssi $|S| < |\mathbf{N}|$; dans le cas contraire (c'est-à-dire lorsque $|S| \geq |\mathbf{N}|$), il est dit *infini*. □

0.63 Exercice. En utilisant la définition 0.62, dites si les ensembles suivants sont finis ou infinis et expliquez pourquoi.

1. C_5
2. $\mathbf{N}$
3. $\mathbf{R}$

□

Remarquez que le fait que $\mathbf{N}$ soit le plus petit ensemble infini est une simple conséquence de la définition 0.62. Remarquez aussi qu'il existe beaucoup d'autres ensembles infinis « aussi petits » que $\mathbf{N}$ (voir exercice 0.60, partie 2).

0.3.3 Ensembles dénombrables et non dénombrables

0.64 Définition. Un ensemble S est dit *dénombrable* ssi $|S| \leq |\mathbf{N}|$. Dans le cas contraire, il est dit *non dénombrable*. □

0.65 Exercice. Dites si les ensembles suivants sont dénombrables ou non dénombrables et expliquez pourquoi.

1. C_5
2. $\mathbf{N}$
3. Un ensemble fini S quelconque

□

Lorsqu'un ensemble est dénombrable, il devient possible de donner une méthode pour énumérer ses éléments de sorte que n'importe quel élément donné soit nommé après un nombre fini d'étapes. Lorsque l'ensemble est fini, il est facile de donner une telle méthode. Voici un exemple simple dans le cas de l'ensemble infini

$$T = \{k \in \mathbf{N} : (\exists n \in \mathbf{N} : k = 3n)\}$$

(l'ensemble des naturels divisibles par 3). La fonction $f : \mathbf{N} \to T$ définie par $f(n) = 3n$ (ou encore par $f = \{(m, n) : n = 3m\}$) est bijective ; par conséquent, $|T| = |\mathbf{N}|$. Pour énumérer les éléments de T, il suffit de le faire dans l'ordre

$$f(0), f(1), f(2), f(3), \ldots,$$

c'est-à-dire

$$0, 3, 6, 9, \ldots$$

Ainsi, n'importe quel nombre naturel *donné*, divisible par 3, sera éventuellement nommé.

> **ATTENTION** : On ne dit pas que *tous* les nombres naturels divisibles par 3 seront éventuellement nommés. Cela ne peut se faire en un temps fini.

Il est parfois plus facile de donner une méthode pour énumérer un ensemble S infini dénombrable que de trouver une fonction bijective $f : \mathbf{N} \to S$.

0.66 Exemple. Montrons que $\mathbf{N} \times \mathbf{N}$ est dénombrable. C'est l'ensemble des paires de nombres naturels. La méthode pour énumérer les paires est la suivante.

1. Énumérer les paires dont la somme des composantes est 0, puis 1, puis 2, ...
2. Pour énumérer les paires dont la somme est n, commencer par la paire dont la première composante est 0, puis 1, puis 2, ...

Le début de l'énumération est donné dans la table suivante, en la lisant de haut en bas et de gauche à droite.

Somme					
0	(0,0)				
1	(0,1)	(1,0)			
2	(0,2)	(1,1)	(2,0)		
3	(0,3)	(1,2)	(2,1)	(3,0)	
4	(0,4)	(1,3)	(2,2)	(3,1)	(4,0)
⋮	⋮	⋮	⋮	⋮	⋮

Une telle description est parfaitement convaincante. Étant donnée n'importe quelle paire (par exemple $(101, 3456)$), il est évident qu'elle sera éventuellement nommée.

Il est plus difficile de trouver une fonction bijective $f : \mathbf{N} \to \mathbf{N} \times \mathbf{N}$. □

Nous savons ce qu'est un ensemble non dénombrable, mais nous ne savons pas encore s'il en existe un. Le prochain théorème nous permettra de montrer que c'est le cas. Il démontre que pour n'importe quel ensemble S, on a $|S| < |\wp(S)|$. En fait, pour un ensemble fini E, cela découle directement des exercices 0.53 et 0.57. En effet,

si $|E| = n$, on a $|\wp(E)| = 2^n$. Comme $n < 2^n$, pour tout $n \in \mathbf{N}$, on a le résultat annoncé (pour les ensembles finis).

Voici maintenant le théorème qui démontre le cas général.

0.67 Théorème. *Soit S un ensemble quelconque (fini ou infini). On a $|S| < |\wp(S)|$.*

DÉMONSTRATION. Évidemment, $|S| \leq |\wp(S)|$, car la fonction $g : S \to \wp(S)$ définie par

$$g(s) = \{s\} \qquad \text{(pour tout } s \in S\text{)}$$

est injective. Il suffit donc de montrer que $|S| \neq |\wp(S)|$. Pour cela, montrons qu'il n'existe pas de fonction surjective de S dans $\wp(S)$.

Procédons par contradiction et supposons qu'il existe une fonction surjective

$$f : S \to \wp(S).$$

Définissons l'ensemble

$$T = \{s \in S : s \notin f(s)\}.$$

> Voici un exemple ; il ne fait pas partie de la preuve. Supposons $S = \mathbf{N}$, $f(3) = \{2, 3, 4\}$ et $f(4) = \{8, 9, 10, 11, 12, 13\}$. On a $3 \in f(3)$ et $4 \notin f(4)$, d'où $3 \notin T$ et $4 \in T$. Fin de l'exemple.

Supposons $s \in S$. Par définition de T, on a

$$s \in T \Leftrightarrow s \notin f(s).$$

Remarquons aussi que $T \subseteq S$ et donc $T \in \wp(S)$. Ces deux propriétés seront utilisées dans la preuve. Dérivons maintenant la contradiction annoncée.

f surjective

$\Rightarrow$ $\langle$ f surjective signifie qu'il est possible d'atteindre n'importe quel élément de $\wp(S)$ (et donc, en particulier, T) à partir d'un élément de S. $\rangle$

$\exists s \in S : f(s) = T$

$\Leftrightarrow$ $\langle$ $s \in f(s) \vee s \notin f(s)$ est vrai ; $(\mathsf{vrai} \wedge f(s) = T) \Leftrightarrow f(s) = T$. Cela peut se vérifier facilement avec les tables de vérité. $\rangle$

$\exists s \in S : (s \in f(s) \vee s \notin f(s)) \wedge f(s) = T$

$\Leftrightarrow$ $\langle$ Distributivité de $\wedge$ sur $\vee$. Encore une fois, cela se vérifie facilement au moyen des tables de vérité. $\rangle$

$\exists s \in S : (s \in f(s) \wedge f(s) = T) \vee (s \notin f(s) \wedge f(s) = T)$

$\Leftrightarrow$ $\langle$ $P \wedge P \Leftrightarrow P$ pour tout prédicat P $\rangle$

$$
\begin{aligned}
& \exists s \in S : (s \in f(s) \wedge s \in f(s) \wedge f(s) = T) \\
& \qquad \vee (s \notin f(s) \wedge s \notin f(s) \wedge f(s) = T) \\
\Leftrightarrow \quad & \qquad \langle \text{ Puisque } f(s) = T,\ s \in f(s) \Leftrightarrow s \in T \text{ et } s \notin f(s) \Leftrightarrow s \notin T. \rangle \\
& \exists s \in S : (s \in f(s) \wedge s \in T \wedge f(s) = T) \vee (s \notin f(s) \wedge s \notin T \wedge f(s) = T) \\
\Rightarrow \quad & \qquad \langle\ P \wedge Q \Rightarrow P\,; \\
& \qquad\ (P \Rightarrow P') \wedge (Q \Rightarrow Q') \Rightarrow (P \vee Q \Rightarrow P' \vee Q')\,; \\
& \qquad\ (P \Rightarrow P') \Rightarrow ((\exists x : P) \Rightarrow (\exists x : P'))\ \rangle \\
& \exists s \in S : (s \in f(s) \wedge s \in T) \vee (s \notin f(s) \wedge s \notin T) \\
\Leftrightarrow \quad & \qquad \langle \text{ Par définition de } T,\ s \in f(s) \Leftrightarrow s \notin T. \rangle \\
& \exists s \in S : (s \notin T \wedge s \in T) \vee (s \in T \wedge s \notin T) \\
\Leftrightarrow \quad & \qquad \langle\ P \wedge \neg P \Leftrightarrow \mathsf{faux}. \rangle \\
& \exists s \in S : \mathsf{faux} \vee \mathsf{faux} \\
\Leftrightarrow \quad & \qquad \langle \text{ Table de vérité. } \rangle \\
& \exists s \in S : \mathsf{faux} \\
\Leftrightarrow \quad & \qquad \langle \text{ La notation } (\exists s \in S : p(s)) \text{ signifie qu'il existe un } s \text{ dans } S \text{ tel} \\
& \qquad \text{que } p(s) \text{ est vrai. Or il ne peut y avoir de } s \text{ tel que } \mathsf{faux} \text{ soit vrai. } \rangle \\
& \mathsf{faux}.
\end{aligned}
$$

À cause de cette contradiction, il faut rejeter l'hypothèse initiale. Il n'y a donc pas de fonction surjective de S dans $\wp(S)$, d'où $|S| < |\wp(S)|$. □

0.68 Corollaire. $|\mathbf{N}| < |\wp(\mathbf{N})|$.

DÉMONSTRATION. C'est une conséquence immédiate du théorème 0.67. □

Ce corollaire montre qu'il y a des ensembles infinis de cardinalités différentes.

Et alors ?

Ce résultat est d'une importance capitale pour le cours. C'est lui qui nous permettra de montrer qu'il y a des problèmes qui ne peuvent être résolus par aucun ordinateur. Montrons immédiatement que votre langage de programmation favori est limité.

0.69 Théorème. Il existe une fonction que votre langage de programmation favori ne peut calculer.

DÉMONSTRATION. Les données d'entrée et de sortie d'un programme sont des séquences de bits. On peut considérer une séquence de bits comme un nombre naturel

exprimé en binaire (en ajoutant un bit 1 au début de la séquence, de sorte que les 0 initiaux soient significatifs). Donc un programme calcule une fonction de $\mathbf{N}$ dans $\mathbf{N}$.

Combien y a-t-il de fonctions de $\mathbf{N}$ dans $\mathbf{N}$? Soient

- $F = \{f : (f : \mathbf{N} \to \{0,1\})\}$ l'ensemble des fonctions de $\mathbf{N}$ dans $\{0,1\}$;
- $G = \{f : (f : \mathbf{N} \to \mathbf{N})\}$ l'ensemble des fonctions de $\mathbf{N}$ dans $\mathbf{N}$.

Sûrement, $|F| \leq |G|$. Mais on peut faire correspondre à chaque fonction $f \in F$ un sous-ensemble $X \subseteq \mathbf{N}$ de la manière suivante :

$$\begin{cases} n \in X \text{ si } f(n) = 1 \\ n \notin X \text{ si } f(n) = 0 \end{cases}$$

(f s'appelle la *fonction caractéristique* de X). Par exemple, la fonction f suivante

n	0	1	2	3	4	5	6	7	8	...
$f(n)$	1	1	0	1	0	0	0	1	1	...

correspond au sous-ensemble

$$X = \{0, 1, 3, 7, 8, \ldots\}.$$

On a donc

$$|F| = |\wp(\mathbf{N})|,$$

d'où

$$|G| \geq |\wp(\mathbf{N})|.$$

G est donc non dénombrable. Mais l'ensemble des programmes qu'on peut écrire avec votre langage préféré est dénombrable, car ces programmes sont construits à partir d'un nombre fini de symboles. Il est donc possible d'énumérer les programmes de longueur 1, puis ceux de longueur 2, etc. (par exemple en ordre alphabétique).

Par conséquent, il existe une fonction que ne peut calculer un langage de programmation donné. □

Cette démonstration fait le lien entre les fonctions de $\mathbf{N}$ dans $\{0,1\}$ et $\wp(\mathbf{N})$. Il y a aussi un lien entre ces fonctions et $\mathbf{R}$. Par exemple, soit $f : \mathbf{N} \to \{0,1\}$ la fonction suivante :

n	0	1	2	3	4	5	6	7	8	...
$f(n)$	1	1	0	1	0	0	0	1	1	...

En ajoutant un « . » avant le premier chiffre, la deuxième ligne peut être considérée comme l'expansion binaire d'un nombre réel $r \in [0,1] \stackrel{\text{déf}}{=} \{x \in \mathbf{R} : 0 \leq x \leq 1\}$. On a

$$|F| = |\,[0,1]\,|.$$

0.70 Exercice. L'ensemble $\mathbf{R}$ est-il dénombrable ? Pourquoi ?

□

0.3.4 Problèmes

1. (a) Comparez les cardinalités des ensembles suivants et justifiez votre réponse.
$$P = \{k \in \mathbf{N} : k < 15 \wedge (\exists j \in \mathbf{N} : k = 2j)\}$$
$$Q = \{k \in \mathbf{N} : k < 15 \wedge (\exists j \in \mathbf{N} : k = 3j)\}$$
   (b) Quelle est la cardinalité des ensembles suivants ? Justifiez votre réponse.
      i. $\{a\}$
      ii. $\emptyset$
   (c) Soient les ensembles $A = \{\{0,1\},\{a\}\}, B = \{\alpha,\beta,\gamma\}$ et $C = \{0,1,2,3,4\}$. Quelle est la cardinalité des ensembles suivants ?
      i. $B \times C$
      ii. $C \times B$
      iii. $A \times B \times C$
      iv. $C \times \emptyset$
2. Donnez un exemple de deux ensembles A et B, *non vides* et *disjoints*, tels que
   (a) $|A| < |B| < |A \cup B|$;
   (b) $|A| < |B| = |A \cup B|$;
   (c) $|A| = |B| = |A \cup B|$.
3. Montrez que $|\mathbf{N}| = |\mathbf{N} \times \mathbf{N} \times \mathbf{N}|$.
4. En utilisant les définitions 0.62 et 0.64, dites si les ensembles suivants sont finis, infinis, dénombrables ou non dénombrables et expliquez pourquoi.
   (a) $\emptyset$
   (b) $P = \{k \in \mathbf{N} : (\exists j \in \mathbf{N} : k = 2j)\}$
5. Donnez des exemples qui montrent que l'intersection de deux ensembles non dénombrables peut être :
   (a) finie (un exemple) ;
   (b) infinie, mais dénombrable (un exemple) ;
   (c) non dénombrable (un exemple).
6. Montrez que si X est un ensemble non dénombrable et Y un ensemble dénombrable, alors $X - Y$ doit être non dénombrable.
7. Montrez que $|\mathbf{R}| \leq |\,[0,1]\,|$.

Chapitre 1

Automates finis et langages réguliers

OBJECTIFS GÉNÉRAUX

1. Comprendre la notion de langage.
2. Comprendre les capacités et les limites des automates finis en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires régulières et les expressions régulières.

1.1 Notion de langage

OBJECTIFS SPÉCIFIQUES

1. Dire si une séquence donnée appartient à un langage donné et expliquer pourquoi.
2. Décrire en français un langage exprimé formellement par une expression ensembliste.
3. Donner une expression ensembliste représentant un langage décrit par une phrase française.

Dans ce chapitre et aussi dans les suivants, nous allons étudier le concept de langage. Tous les concepts que nous introduirons essaieront de mieux cerner les propriétés des langages. Étant donnés nos objectifs d'ordre informatique, les langages que nous examinerons seront exprimés par des représentations qu'on appellera, dépendant du contexte : automates, grammaires, expressions ou algorithmes.

Dans les langages naturels comme le français, on distingue, entre autres, trois entités : les lettres, les mots et les phrases. Un ensemble de lettres agencées côte à côte forme un mot. Parallèlement, un ensemble de mots unis en chaîne de gauche à droite forme une phrase. Ce ne sont pas toutes les collections de lettres qui forment des mots valides et de même, ce ne sont pas tous les agencements de mots qui forment des phrases acceptables. Cette situation est vraie aussi pour les langages informatiques comme Pascal ou C. Seulement certaines chaînes de caractères forment des commandes reconnaissables.

Il a été nécessaire d'adopter une structure des langages dans laquelle la décision d'accepter ou de refuser un mot ou une phrase n'est pas faite à l'aveuglette mais est basée sur des règles non ambiguës. C'est la structure théorique de ces règles que nous allons étudier. Nous obtiendrons des machines théoriques qui pourront servir à vérifier si oui ou non, un mot appartient bien à un langage ; ce sera le cas des automates en particulier. Il y aura d'autres représentations de langages, par exemple : grammaires et expressions régulières. Nous classerons les différents langages découverts. Plus nous avancerons dans le cours, plus les classes de langages s'agrandiront et engloberont les précédentes. Ainsi nous connaîtrons mieux nos limites dans l'expression et la vérification des langages informatiques.

L'application la plus directe de cette étude des langages est la construction des compilateurs. Une des fonctions des compilateurs est de vérifier si les mots d'un programme respectent les règles de formation imposées par le langage. Par exemple, en Pascal, le compilateur doit reconnaître les termes : PROGRAM, BEGIN et les instructions de la forme IF ...THEN ...ELSE, mais doit refuser la suite de mots

IF ...WHILE ...THEN. Chaque compilateur doit donc être capable de dire si un programme est syntaxiquement correct. La partie du compilateur qui s'en occupe se nomme analyseur syntaxique. L'analyseur syntaxique n'est pas simple à construire. Il suffit de penser à la syntaxe du langage Pascal pour comprendre la difficulté qu'ont eu ceux qui ont réalisé son analyseur syntaxique. C'est cette complexité des langages qui en a motivé l'étude théorique et rigoureuse.

Pour débuter notre étude des langages, nous devons nous entendre sur ce qu'on appelle langage. Jusqu'ici, nous prenions pour acquis la définition informelle, commune à tous. Or, les termes que nous utiliserons doivent être clairs et bien définis. Commençons par le plus simple, ce qu'est un alphabet.

1.1.1 Alphabets

1.1 Définition. Un *alphabet* est un ensemble fini, non vide, de symboles. On le dénote généralement par Σ. □

1.2 Exemple. $\Sigma = \{a, b, c\}$ est un alphabet. □

1.3 Exercice. L'ensemble des caractères ASCII est-il un alphabet ? □

Nous dirons que *chaîne*, *séquence* et *suite* sont synonymes, de même que *caractère* et *symbole*. Ainsi, une chaîne finie de caractères est la même chose qu'une suite finie de caractères ou qu'une séquence finie de symboles.

1.4 Notation. La séquence ne contenant aucun symbole, appelée séquence vide, est aussi une séquence de symboles. Nous la dénoterons par λ. Puisque la concaténation d'une séquence vide et d'une séquence quelconque s est la séquence s elle-même, la séquence vide obéit aux lois suivantes :

$$\lambda s = s\lambda = s.$$

□

1.5 Exercice. Simplifiez l'expression des séquences suivantes.

1. $\lambda ab\lambda c =$
2. $\lambda\lambda =$

□

1.6 Définition. Soit Σ un alphabet. On dénote par Σ^* l'ensemble de toutes les séquences finies de symboles de Σ. □

Notez que, d'après cette définition, la séquence vide appartient à Σ^*, pour tout alphabet Σ.

1.7 Exemple. Voici un alphabet $\Sigma = \{a, b, c\}$. Alors

$$\Sigma^* = \{\lambda, a, b, c, aa, ab, ac, \ldots, aaa, aab, \ldots, aaaa, aaab, \ldots, aaaaa, \ldots\}.$$

Notons que le symbole λ n'appartient pas à l'alphabet Σ. Il représente la séquence vide. □

1.8 Remarque. Le symbole * représente un opérateur unaire[1] qui, appliqué à un alphabet Σ, donne un ensemble infini Σ^*. On dit que Σ^* est la *fermeture* de l'alphabet Σ. Bien que la fermeture d'un alphabet soit un ensemble infini, il est néanmoins dénombrable. Il suffit d'énumérer les séquences de longueur 0, puis 1, puis 2, ... □

1.9 Exercice. Soit l'alphabet $\Sigma = \{0, 1\}$. Énumérez les 15 premières séquences (par ordre de longueur croissante) de Σ^*.

□

1.10 Exercice. Supposons $a \in \Sigma$. La séquence contenant une infinité de a, c'est-à-dire $aaaaaa\ldots$, appartient-elle à Σ^* ?

□

1.11 Notation. Soient Σ un alphabet quelconque, $s \in \Sigma^*$ et $n \in \mathbf{N}$. Alors

$$s^n = \underbrace{s \ldots s}_{n \text{ fois}}.$$

□

1.12 Exemple.

- $a^4 = aaaa$
- $(mn)^2 t^3 = mnmnttt$

□

1.13 Exercice. Exprimez les séquences suivantes sans utiliser d'exposant. L'alphabet est $\Sigma = \{0, 1\}$.

1. $0^2 1^2$
2. $(01)^2$
3. $((00)^2 1^3)^2$

[1] On appelle cet opérateur l'*étoile de Kleene*, d'après le nom du premier mathématicien qui a utilisé cette notation.

4. 1^0

□

Les exposants permettent de décrire certains ensembles de manière concise.

1.14 Exemple. L'ensemble $\{a^m b a^n : m \in \mathbf{N} \wedge n \in \mathbf{N}\}$ est l'ensemble des séquences ayant un seul b, précédé et suivi d'un nombre quelconque de a. □

1.15 Exercice. Décrivez en mots les ensembles L_1 et L_2 suivants. Dites lesquelles des séquences données appartiennent à l'ensemble et expliquez pourquoi.

1. $\Sigma = \{a\}, \quad L_1 = \{a^{n^2} : n \in \mathbf{N}\}$
   - a
   - $aaaaaa$
2. $\Sigma = \{0, 1, +, -\}, \quad L_2 = \{s^j t s^j : j \in \mathbf{N}^+ \wedge (s = 0 \vee s = 1) \wedge (t = + \vee t = -)\}$
   - $0 + +0$
   - $0 + 10$

□

1.16 Exercice. Donnez une expression ensembliste décrivant l'ensemble des séquences sur l'alphabet $\{a, b\}$ qui contiennent deux a, l'un au début et l'autre à la fin, et qui contiennent un nombre pair de b entre les deux a.

□

1.17 Définition. La *longueur* d'une séquence finie w est le nombre de symboles qu'elle contient. On écrit $|w|$ pour représenter cette longueur. □

1.18 Exemple. Soit $\Sigma = \{a, b, c\}$. On a $|a| = 1$, $|abc| = 3$ et $|aaa| = 3$. □

1.19 Exercice. Pour l'alphabet $\Sigma = \{a, b, c\}$, donnez la valeur des expressions suivantes.

1. $|\lambda|$
2. $|aa\lambda bca\lambda|$
3. $|(a^2b^3c^4)^2|$

□

On peut trouver une certaine similarité entre Σ^* et $\wp(\Sigma)$ mais il ne faut pas confondre l'un avec l'autre. Ce sont deux entités bien différentes. En particulier,

$$\wp(\{a\}) = \{\emptyset, \{a\}\}$$

est fini mais

$$\{a\}^* = \{\lambda, a, aa, aaa, aaaa, \ldots\}$$

est infini.

1.1.2 Langages

1.20 Définition. Un *langage* sur un alphabet Σ est un sous-ensemble de l'ensemble Σ^*. □

1.21 Exemple. Soit l'alphabet $\Sigma = \{a, b, c\}$. Les ensembles suivants sont des langages sur Σ : $L = \{a, b, aa, abc, abbc\}$, $P = \{\lambda, a, aa, aaa, \ldots\}$. En effet, $L \subseteq \Sigma^*$ et $P \subseteq \Sigma^*$. □

Un langage peut être fini ou infini et il peut contenir la séquence vide ou ne pas la contenir. Il faut aussi se rappeler que, quel que soit le langage, les séquences qu'il contient sont toujours de longueur finie.

1.22 Remarque. D'après la définition 1.20, l'ensemble des langages sur un alphabet Σ est $\wp(\Sigma^*)$. Comme Σ^* est infini et que $|\Sigma^*| < |\wp(\Sigma^*)|$ (d'après le théorème 0.67), on voit que l'ensemble des langages sur un alphabet donné est non dénombrable. Rappelons que Σ^* est dénombrable, pour tout alphabet Σ. □

1.23 Exercice. Dites si ces affirmations sont vraies ou fausses. Expliquez votre choix.

1. $\{0, 1, +, -, \times, \div\}$ est un alphabet, où $0, 1, +, -, \times$ et $\div$ sont des symboles.
2. $\{a_i : i \in \mathbf{N}\}$ est un alphabet, où chaque a_i est un symbole et $a_i \neq a_j$ si $i \neq j$.
3. λ est un langage.
4. $\emptyset$ est un langage.
5. $\{\lambda\}$ est un langage.
6. $\emptyset \in \{\lambda\}$.
7. $\lambda = \emptyset$.

□

1.1.3 Diagrammes de transitions

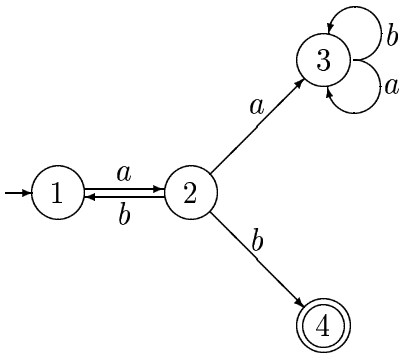
Nous venons de voir ce qu'est un langage et comment on peut définir un langage au moyen d'une expression ensembliste. Dans cette section, nous introduisons le concept de diagramme de transitions, qui nous permettra d'aboutir à une vision opérationnelle de ce qu'est un langage. En effet, les diagrammes de transitions sont des composantes essentielles des dispositifs de reconnaissance de langages (automates, machines de Turing) qui sont présentés par la suite.

1.24 Définition. Un *diagramme de transitions* est une collection finie d'*états* (représentés par des cercles) et de *transitions* (représentées par des arcs dirigés) telle que :

- Chaque état *peut* être étiqueté. Les étiquettes sont uniques.
- Un et un seul des états est pointé par une petite flèche ($\rightarrow$) qui l'identifie comme *état initial*.
- Certains cercles peuvent être doubles (c'est-à-dire formés de deux cercles concentriques). Ce sont les *états finaux* (ou états *accepteurs*). Il peut ne pas y avoir d'état final.
- Chaque transition relie deux états (pas nécessairement différents).
- Chaque transition est étiquetée par un symbole d'un alphabet.

Pour éviter d'avoir à tracer un trop grand nombre de transitions, certaines abréviations sont utilisées. Celles-ci permettent d'étiqueter une transition par plusieurs symboles, ou bien par une séquence de symboles ou encore par un symbole spécial. Voyez la notation 1.26. □

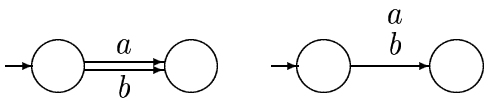
1.25 Exemple. Voici un exemple d'un diagramme de transitions associé à l'alphabet $\Sigma = \{a, b\}$. On remarque toutes les caractéristiques précisées. L'ensemble des états est $\{1, 2, 3, 4\}$. Il y a un et un seul état initial, l'état 1 (la petite flèche l'indique). Chaque arc est étiqueté. On remarque aussi qu'il y a un état final, l'état 4.



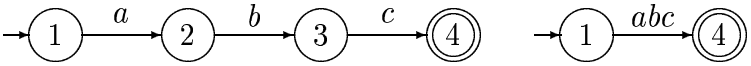
□

1.26 Notation. Pour éviter de surcharger un diagramme, il est possible d'utiliser certaines abréviations.

1. Au lieu d'utiliser plusieurs arcs entre deux états, on en utilise un seul en l'étiquetant avec les étiquettes des arcs originaux empilées l'une au-dessus de l'autre. Ainsi, le diagramme de droite est une abréviation du diagramme de gauche.



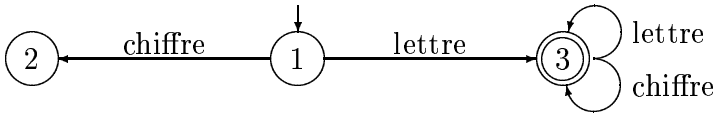
2. Dans le diagramme de droite ci-dessous, l'étiquette abc est une séquence de symboles. Ce diagramme est une abréviation du diagramme de gauche.



3. Le diagramme suivant est un diagramme de transitions associé à l'alphabet $\Sigma =$ *ensemble des caractères ASCII*. Ses transitions sont étiquetées par des symboles spéciaux (lettre, chiffre) représentant chacun un certain sous-ensemble de Σ :

$$\text{lettre} = \{a, \ldots, z, A, \ldots, Z\} \quad \text{et} \quad \text{chiffre} = \{0, \ldots, 9\}.$$

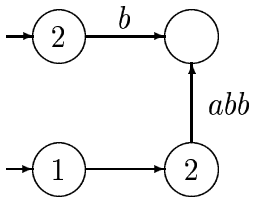
Sans cette abréviation, il faudrait tracer $26 + 26 + 10 = 62$ arcs de l'état 3 à l'état 3.



Ce diagramme représente la syntaxe d'un identificateur telle qu'elle est définie dans plusieurs langages de programmation. En effet, si on suit les transitions en partant de l'état initial 1, jusqu'à l'état final 3 (où on peut boucler), on obtient une description d'un identificateur en concaténant les étiquettes rencontrées sur les transitions.

□

1.27 Exercice. La figure ci-dessous est un diagramme de transitions associé à l'alphabet $\Sigma = \{a, b\}$, et n'ayant pas toutes les caractéristiques exigées. Énumérez toutes celles qui ne sont pas respectées.



□

Voyons maintenant comment construire une représentation tabulaire d'un diagramme de transitions dans le cas où il y a au plus une transition étiquetée t qui quitte un état donné s (le diagramme de l'exemple 1.25 ne satisfait pas cette contrainte, puisque deux transitions étiquetées b quittent l'état 2). Cette représentation pourra être utilisée par un algorithme.

1.28 Définition. Une *table de transitions* associée à un diagramme de transitions est une matrice à deux dimensions telle que :

- La matrice a une ligne pour chaque état du diagramme.
- Elle a une colonne pour chaque symbole utilisé comme étiquette d'un arc du diagramme.
- L'entrée de la $n$ième ligne et de la $m$ième colonne est l'état atteint dans le diagramme de transitions en quittant l'état n par l'arc dont l'étiquette est celle de la colonne m. Si un tel état n'existe pas, l'entrée contient le mot ERREUR.
- On ajoute une colonne indicée par le caractère spécial de fin de séquence FIN. Une entrée de cette colonne est le mot ACCEPTE si l'état (ligne) est final ou ERREUR sinon.

□

1.29 Exemple. La table de transitions ci-dessous correspond au dernier diagramme de transitions de la notation 1.26. Les lignes correspondent aux états et les colonnes aux symboles de l'alphabet. Il y a une colonne supplémentaire étiquetée FIN ; elle indique ce qu'il faut faire si la fin de la séquence lue est atteinte. Voici un exemple de lecture d'une entrée dans la table. L'obtention d'une lettre dans l'état 3 force une transition vers l'état 3, comme on peut le voir à la ligne 3, colonne « lettre ».

	lettre	chiffre	FIN
1	3	2	ERREUR
2	ERREUR	ERREUR	ERREUR
3	3	3	ACCEPTE

□

1.30 Exercice. D'après la table de transitions de l'exemple 1.29, quelle transition doit être effectuée à l'obtention d'un chiffre dans l'état 2 ?

□

La figure 1.1 donne l'analyseur lexical (sous forme algorithmique) venant de la table de transitions de l'exemple 1.29. Il simule le comportement du diagramme de transitions en utilisant la table de transitions qui lui est associée pour analyser une séquence de symboles dont la lecture se fait symbole par symbole.

Cet algorithme peut facilement se généraliser à des diagrammes de transitions plus complexes. Lorsqu'il termine dans l'état ACCEPTE, on dit que la séquence d'entrée est acceptée ou reconnue. Une séquence acceptée correspond à une suite d'étiquettes menant de l'état initial à un état final.

État := 1 ;
Répéter
  Lire le symbole d'entrée suivant et le mettre dans la variable Symbole ;
  Dans le cas où Symbole est
    une lettre : Lu := "lettre" ;
    un chiffre : Lu := "chiffre" ;
    FIN : Lu := "FIN" ;
    Autre : État := "ERREUR" ;
  Si État ≠ "ERREUR" alors État := Table [État, Lu] ;
jusqu'à ce que État = "ACCEPTE" ou État = "ERREUR".

FIG. 1.1 – Analyseur lexical correspondant à la table de l'exemple 1.29

1.1.4 Problèmes

1. L'ensemble vide $\emptyset$ est-il un alphabet ?
2. Décrivez en mots l'ensemble L suivant. Dites lesquelles des séquences données appartiennent à l'ensemble et expliquez pourquoi.

$$\Sigma = \{t, o\}, \quad L_1 = \{(to)^{2n} : n \in \mathbf{N}\}$$

   - *ttoo*
   - *tototo*

3. Donnez une expression ensembliste décrivant l'ensemble des séquences sur l'alphabet $\{0\}$ qui contiennent un nombre premier de 0.
4. Dites si les affirmations suivantes sont vraies ou fausses. Expliquez votre choix.

   (a) $\emptyset \not\subseteq \{\lambda\}$.

   (b) $\lambda \in \emptyset$.

   (c) $\lambda \subseteq \lambda$.

   (d) $\emptyset = \{\lambda\}$.

   (e) Si L est un langage sur Σ, alors $L \in \wp(\Sigma^*)$.

   (f) La chaîne $\lambda a \lambda \lambda a$ est un élément du langage $\{a, aa, aaa\}$.

   (g) Si un langage a pour représentation ensembliste $\{\lambda, \lambda\lambda, \lambda\lambda\lambda\}$ alors cet ensemble contient trois séquences de symboles.

   (h) Σ^* est un langage, pour tout alphabet Σ.

   (i) Soit L un langage. Alors $\lambda \in L$.

   (j) $\Sigma^* - \Sigma^*$ est un langage.

5. Construisez l'analyseur lexical correspondant au diagramme de transitions de l'exemple 1.32.
6. Construisez un diagramme de transitions qui reconnaît des expressions arithmétiques de longueur arbitraire. Les expressions sont constituées d'entiers positifs (sans le symbole du signe) séparés par les symboles d'addition, de soustraction, de multiplication ou de division.
7. À partir du diagramme précédemment construit, donnez la table de transitions et son analyseur lexical.

1.2 Automates finis déterministes

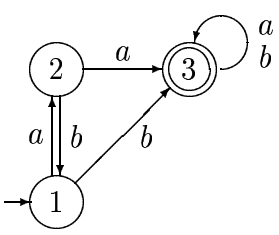
OBJECTIFS SPÉCIFIQUES

1. Démontrer le fonctionnement d'un automate fini déterministe donné en énumérant les configurations du ruban et les états consécutifs à une configuration initiale donnée.
2. Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate fini déterministe donné.

Maintenant que nous avons introduit le concept de langage, nous allons nous attaquer plus profondément à la question de la représentation des langages par des diagrammes de transitions. La notion d'automate fini déterministe présentée ci-dessous formalise ces derniers. Auparavant, examinons des types particuliers de diagrammes de transitions.

1.31 Définition. Un diagramme de transitions avec alphabet Σ est dit *complètement défini* si, pour chaque symbole $s \in \Sigma$ et chaque état e, il y a au moins une transition étiquetée s qui quitte e. □

1.32 Exemple. Le diagramme de transitions ci-dessous, associé à l'alphabet $\Sigma = \{a, b\}$, est complètement défini car, pour chacun des états 1, 2 et 3, il y a au moins une transition étiquetée a et au moins une transition étiquetée b qui quitte l'état en question.



□

1.33 Exercice. Le diagramme de transitions associé à l'alphabet $\Sigma = \{a, b\}$ représenté à l'exemple 1.25 est-il complètement défini ?

□

1.34 Définition. Un diagramme de transitions avec alphabet Σ est dit *non ambigu* si, pour chaque état e et chaque symbole $s \in \Sigma$, il existe au plus une transition quittant e et étiquetée s. □

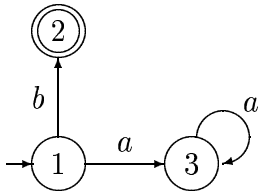
1.35 Exemple. Le diagramme de transitions de l'exemple 1.25 est ambigu car, de l'état 2, il y a une transition étiquetée b qui se dirige vers l'état 1 et il y a une transition de même étiquette qui se rend vers l'état 4. □

1.36 Exercice. Le diagramme de transitions associé à l'alphabet $\Sigma = \{a, b\}$ représenté à l'exemple 1.32 est-il ambigu ?

□

1.37 Définition. Un diagramme de transitions *déterministe* par rapport à un alphabet donné est un diagramme complètement défini et non ambigu. □

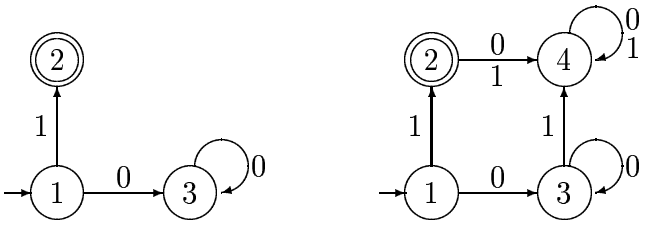
1.38 Exemple. En supposant que son alphabet est $\Sigma = \{a, b\}$, le diagramme de transitions ci-dessous n'est pas déterministe car il n'est pas complètement défini : entre autres, il n'y a pas de transition étiquetée b qui quitte l'état 2. Le diagramme de transitions associé à l'alphabet $\Sigma = \{a, b\}$ de l'exemple 1.25 n'est pas non plus déterministe car il est ambigu (selon l'exemple 1.35).



□

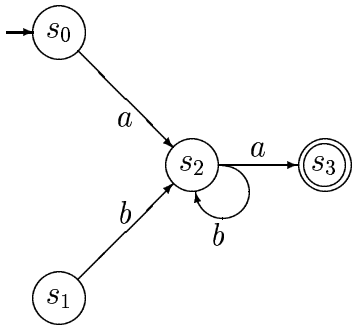
1.39 Remarque. Voici une manière de procéder pour rendre déterministe un diagramme de transitions non ambigu. On commence par ajouter un état non final au diagramme (on appelle cet état un *état poubelle* ou *état puits*, pour des raisons qui deviendront évidentes). Pour chaque symbole de l'alphabet, on dessine un arc qui part de l'état précédemment ajouté et qui se rend à ce même état. Enfin on ajoute des arcs des autres états jusqu'au nouvel état jusqu'à ce que chaque état soit l'origine d'un arc pour chaque symbole de l'alphabet. On remarquera que souvent cette technique alourdit le diagramme et peut en cacher l'essentiel. C'est pour cette raison que certains des diagrammes ultérieurs ne seront pas rendus complètement définis. □

1.40 Exemple. Le diagramme de gauche ci-dessous est non déterministe et non ambigu. On peut lui appliquer la technique précédente pour le rendre déterministe par rapport à l'alphabet $\Sigma = \{0, 1\}$. Il suffit d'ajouter l'état 4 qui recevra des transitions venant des états 2, 3 et 4 de manière à compléter le diagramme pour le rendre complètement défini. Le diagramme résultant, donné à droite, est non ambigu et complètement défini ; il est donc déterministe.



□

1.41 Exercice. Si possible, rendez déterministe le diagramme ci-dessous. L'alphabet est $\Sigma = \{a, b\}$.



□

1.42 Définition. Un *automate fini déterministe* consiste en un quintuple de la forme $(S, \Sigma, \delta, \iota, F)$. Voici la description des différentes composantes.

- S est un ensemble fini d'états.
- Σ est l'alphabet de l'automate.
- δ est une fonction (appelée fonction de transition) de $S \times \Sigma$ dans S.
- $\iota \in S$ est l'état initial.
- $F \subseteq S$ est l'ensemble des états finaux.

□

1.43 Exemple. Le quintuplet $(\{A, B, C\}, \{a, b, c\}, \delta, A, \{A, B\})$, où la fonction

$$\delta : \{A, B, C\} \times \{a, b, c\} \to \{A, B, C\}$$

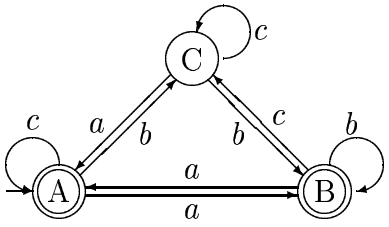
est définie par

$$\begin{array}{lll} \delta(A,a)=B & \delta(A,b)=C & \delta(A,c)=A \\ \delta(B,a)=A & \delta(B,b)=B & \delta(B,c)=C \\ \delta(C,a)=A & \delta(C,b)=B & \delta(C,c)=C, \end{array}$$

est un automate avec

- ensemble d'états $\{A,B,C\}$,
- alphabet $\{a,b,c\}$,
- fonction de transition δ,
- état initial A,
- ensemble d'états finaux $\{A,B\}$.

Le diagramme de transitions correspondant à la fonction de transition δ est le suivant.



Remarque : On peut aussi définir δ comme suit.

$$\begin{aligned} \delta = \{&((A,a),B),((A,b),C),((A,c),A),\\ &((B,a),A),((B,b),B),((B,c),C),\\ &((C,a),A),((C,b),B),((C,c),C)\} \end{aligned}$$

□

1.44 Exercice. Donnez le diagramme de transitions correspondant à l'automate

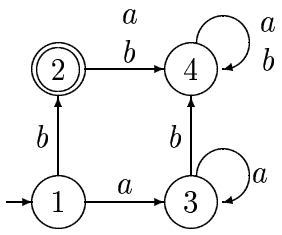
$$(\{0,1,2,3\},\{f,g,h\},\delta,0,\{2\}),$$

où δ est définie par

$$\begin{array}{lll} \delta(0,f)=3 & \delta(0,g)=1 & \delta(0,h)=1 \\ \delta(1,f)=1 & \delta(1,g)=2 & \delta(1,h)=3 \\ \delta(2,f)=0 & \delta(2,g)=3 & \delta(2,h)=3 \\ \delta(3,f)=3 & \delta(3,g)=3 & \delta(3,h)=3. \end{array}$$

□

1.45 Exercice. Soit $\Sigma = \{a, b\}$. Donnez l'automate $(S, \Sigma, \delta, \iota, F)$ correspondant au diagramme de transitions ci-dessous.



□

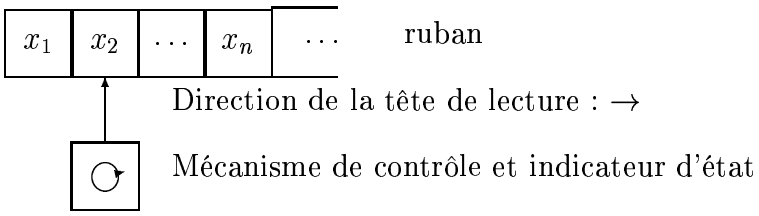
1.46 Remarque. Le diagramme de transitions n'est pas un automate ; il n'est que la représentation de la fonction de transition de la machine. Cependant, nous dirons souvent d'un diagramme de transitions que c'est un automate, à cause du lien direct qu'il y a entre les deux notions.

On remarque les caractéristiques suivantes d'un automate fini déterministe quelconque :

- Pour chaque couple $(p, x) \in S \times \Sigma$, il existe un et un seul $q \in S$ tel que $\delta(p, x) = q$ (par définition même de ce qu'est une fonction).
- À tout automate fini déterministe correspond un diagramme de transitions déterministe. C'est parce que, comme toutes les fonctions, la fonction de transition d'un automate fini déterministe est totale (complètement définie) et non ambiguë.
- Le qualificatif « fini » est utilisé pour indiquer que l'automate ne possède qu'un nombre fini d'états, que son alphabet est fini et que son nombre de transitions est aussi fini.

□

Pour rendre un peu plus concret le concept d'automate, nous pouvons visualiser un automate comme une machine avec un ruban contenant une séquence de caractères (symboles) à analyser, une tête de lecture pour lire les caractères sur le ruban, et un mécanisme de contrôle qui peut changer d'état.

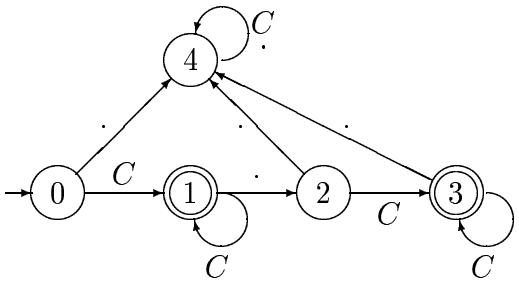


- Le ruban est divisé en cases et est infini vers la droite. La séquence finie à analyser est placée au début du ruban. La tête de lecture repose sur le prochain symbole à lire. Initialement, la tête de lecture est sur la case la plus à gauche.
- L'analyse d'une séquence de symboles se fait séquentiellement. Tous les symboles de la séquence doivent être lus par l'automate, un à la suite de l'autre en partant du premier symbole de gauche.
- Le mécanisme de contrôle a la tâche de choisir un nouvel état, à partir de l'état actuel et du symbole lu (conformément à la fonction de transition). C'est comme un programme qui calcule l'état suivant.
- L'automate accepte (reconnaît) la séquence d'entrée si et seulement si la transition effectuée lors de la lecture du dernier symbole l'amène dans un état final.

1.47 Définition. L'automate fini déterministe $M = (S, \Sigma, \delta, \iota, F)$ *accepte* (ou *reconnaît*) la séquence $x_1x_2x_3 \ldots x_n$ (où $n \in \mathbf{N}$) si, et seulement si, il y a une séquence d'états $s_0, s_1, \ldots, s_n$ tels que $s_0 = \iota, s_n \in F$, et pour tout $j = 1, \ldots, n$, $\delta(s_{j-1}, x_j) = s_j$. Dans le cas contraire, on dit que l'automate *rejette* la séquence. □

En d'autres mots, l'automate accepte une séquence de symboles ssi la séquence, lue de gauche à droite, correspond à une suite d'arcs conduisant de l'état initial à un état final.

1.48 Exemple. Soit l'automate déterministe M décrit par le diagramme de transitions suivant. Son alphabet est $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \cdot\}$. Le symbole C est un symbole spécial remplaçant n'importe quel chiffre : $C \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.



Voici une suite de configurations du ruban et d'états de l'automate décrivant l'acceptation de la séquence 123.45 par M. La tête de lecture est sur le symbole souligné.

ruban	état
123.45	0
123.45	1
123.45	1
123.45	1
123.45	2
123.45	3
123.45	3

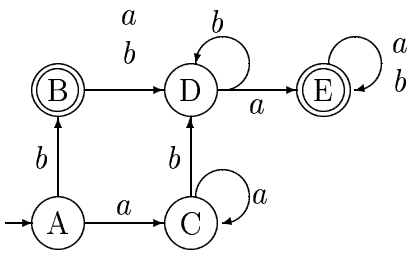
La séquence est acceptée, puisque l'automate est dans un état final (3) après la lecture de la séquence.

La suite de configurations du ruban et d'états de l'automate ci-dessous décrit le rejet de la séquence 88.0. . L'automate rejette la séquence car il termine la lecture dans l'état 4, qui n'est pas un état final.

ruban	état
88.0.	0
88.0.	1
88.0.	1
88.0.	2
88.0.	3
88.0.	4

□

1.49 Exercice. Donnez les configurations du ruban et les états de l'automate fini déterministe ci-dessous lors de la lecture des séquences suivantes. Dites si les séquences sont acceptées.

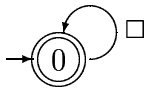


1. a
2. b
3. aab
4. aaababb

□

1.50 Remarque. La définition 1.47 ci-dessus indique aussi comment la séquence vide peut être acceptée. Il suffit de prendre $n = 0$ dans la définition. La séquence d'entrée est alors la séquence vide λ. Pour l'accepter, l'état initial doit aussi être final. Aucune transition n'est faite. □

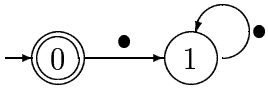
1.51 Exemple. L'automate suivant, dont l'alphabet est $\{\square\}$, accepte la séquence vide λ. En effet, après avoir lu le dernier symbole d'entrée, il est dans un état final. La suite de configurations menant à l'acceptation est donnée à droite de l'automate.

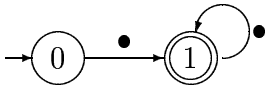


ruban	état
_	0

□

1.52 Exercice. Dites si les automates suivants acceptent λ.

1. 

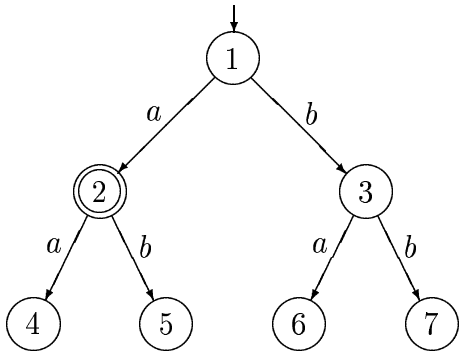
2. 

3. $(\{S,T\},\{+,-\},\{((S,+),S),((S,-),T),((T,+),T),((T,-),S)\},S,\{S,T\})$
4. $(\{S,T\},\{+,-\},\{((S,+),S),((S,-),T),((T,+),T),((T,-),S)\},S,\{T\})$

□

1.2.1 Problème

1. Si possible, rendez déterministe le diagramme ci-dessous. L'alphabet est $\Sigma = \{a,b\}$.



1.3 Les limites des automates déterministes

OBJECTIFS SPÉCIFIQUES

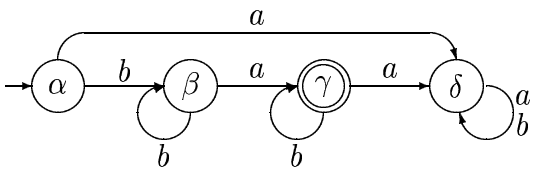
1. Décrire le langage accepté par un automate fini déterministe donné.
2. Construire un automate fini déterministe acceptant un langage donné.
3. Démontrer qu'un langage donné n'est pas régulier.

1.3.1 Langages réguliers

Maintenant, nous allons relier la notion d'automate fini déterministe à celle de langage. Nous verrons qu'un automate accepte un langage bien précis.

1.53 Définition. Soit M un automate fini déterministe qui a pour alphabet Σ. Nous dénotons par $L(M)$ l'ensemble de toutes les séquences de symboles reconnues par l'automate M. Nous dirons que $L(M)$ est le *langage reconnu*(ou *accepté*)par l'automate fini M. □

1.54 Exemple. Soit A l'automate fini déterministe suivant.



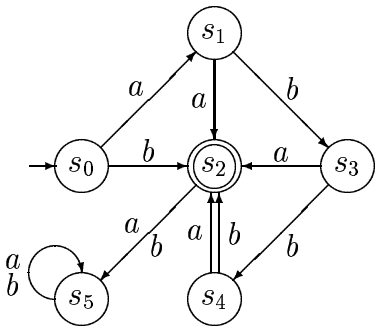
Les séquences acceptées ont un ou plusieurs b, suivis d'un unique a, suivi de zéro ou plusieurs b. On a donc

$$L(A) = \{b^m a b^n : m \in \mathbf{N}^+ \wedge n \in \mathbf{N}\}.$$

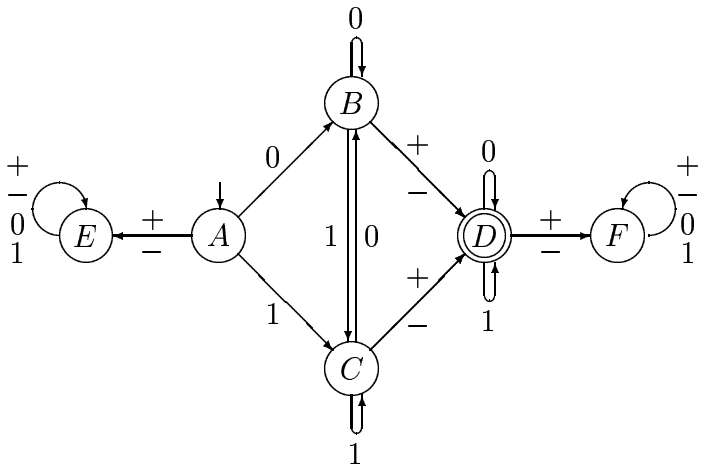
□

1.55 Exercice. Dites, en mots et au moyen d'une expression ensembliste, quel est le langage reconnu par chacun des automates déterministes suivants.

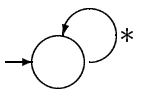
1.



2.



3.



□

1.56 Exercice. Soit l'alphabet $\Sigma = \{b, h, i, o, u\}$. Construisez un automate fini déterministe qui reconnaît le langage demandé.

1. $\{hibou, hi, bou, hou\}$
2. $\{(ho)^n : n \in \mathbf{N}\}$

□

1.57 Définition. Un langage est dit *régulier* s'il existe un automate fini déterministe qui le reconnaît. □

Les langages de l'exercice 1.56 sont donc réguliers. Une façon de montrer qu'un langage est régulier est de construire un automate qui reconnaît ce langage. Nous verrons d'autres méthodes dans les sections suivantes.

1.58 Exercice.

1. Montrez que Σ^* est régulier.
2. Montrez comment on peut transformer un automate fini *déterministe* $M = (S, \Sigma, \delta, \iota, F)$ pour qu'il accepte $\Sigma^* - L(M)$. Cet exercice démontre que le complément d'un langage régulier (relativement à Σ^*) est régulier.

□

1.59 Exercice.

1. Soit $\Sigma = \{a, b\}$. Énumérez tous les automates finis déterministes sur Σ qui ont 1 ou 2 états. Considérez que les états sont étiquetés.
2. Dites combien il y a d'automates finis déterministes à n états étiquetés sur un alphabet Σ tel que $|\Sigma| = k$.

□

On conclut de cet exercice que l'ensemble des automates sur un alphabet Σ est dénombrable. Il suffit d'énumérer les automates à 1 état, puis ceux à 2 états, puis ceux à 3 états, etc.

1.60 Théorème. *Pour tout alphabet Σ, il existe un langage qu'aucun automate fini déterministe ne peut reconnaître.*

DÉMONSTRATION. Nous pouvons considérer seulement les automates dont l'alphabet est Σ, car l'ajout d'un arc avec un symbole $x \notin \Sigma$ ne peut affecter la reconnaissance d'une séquence $w \in \Sigma^*$.

Dénotons par A_Σ l'ensemble des automates déterministes finis sur l'alphabet Σ. D'après l'exercice 1.59, cet ensemble est dénombrable, c'est-à-dire

$$|A_\Sigma| \leq |\mathbf{N}|.$$

D'autre part, selon la remarque 1.22, l'ensemble des langages sur Σ est non dénombrable, c'est-à-dire

$$|\mathbf{N}| < |\wp(\Sigma^*)|.$$

Par conséquent,

$$|A_\Sigma| < |\wp(\Sigma^*)|.$$

□

L'ensemble des langages est donc beaucoup plus grand que celui des automates. Cependant, la preuve ci-dessus n'est pas constructive. Nous savons qu'il existe un langage qui ne peut être reconnu par aucun automate fini, mais nous n'avons pas d'exemple d'un tel langage. La section suivante nous en donnera un.

1.3.2 Langages non réguliers

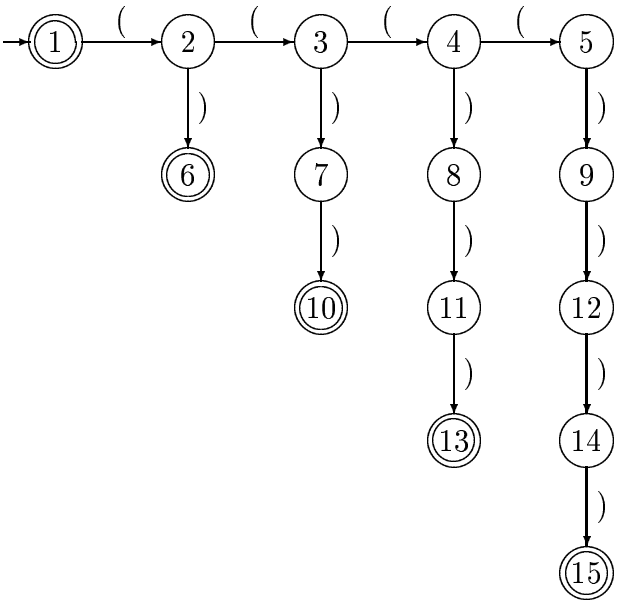
Nous allons montrer qu'aucun automate fini déterministe ne peut déterminer si les parenthèses d'une expression arithmétique sont correctement équilibrées. Par exemple, les deux premières expressions ci-dessous sont incorrectes, alors que la troisième est correcte.

$$a + (b \div c \times (a - d)$$
$$a + b) \div c \times (a - d$$
$$a + (b \div c \times (a - d) + g)$$

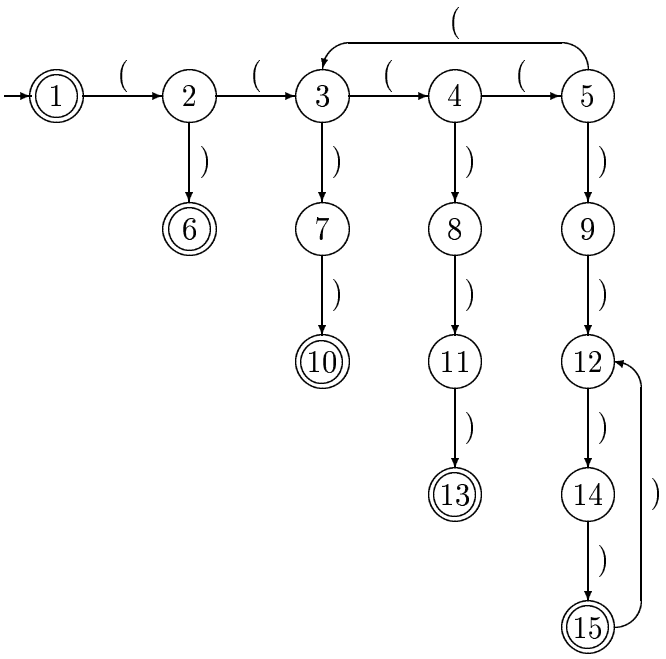
Il est facile de dire si les parenthèses sont correctement équilibrées en lisant l'expression de gauche à droite, et en se rappelant la différence entre le nombre de parenthèses gauches g et le nombre de parenthèses droites d. En tout temps lors de la lecture, on doit avoir $g \geq d$. A la fin de la lecture, il faut que $g = d$. Le problème, c'est qu'un automate fini ne peut faire un tel calcul lorsqu'on lui donne une expression qui est « trop longue ».

En fait, nous allons montrer qu'un automate fini ne peut reconnaître le langage $\{(^n)^n | n \in \mathbf{N}\}$ sur l'alphabet $\{(,)\}$. Ce langage est pourtant plus simple que celui des expressions arithmétiques parenthésées.

Considérons l'automate suivant. Afin de garder seulement les détails essentiels pour la discussion qui va suivre, certaines transitions ont été supprimées. On remarque que le diagramme de transitions n'est pas complètement défini, mais qu'il n'est pas ambigu. On peut donc lui appliquer la technique de la remarque 1.39 afin de le rendre complètement défini si désiré.



Cet automate reconnaît le langage fini $\{\lambda, (), (()), ((())), (((())))\}$. Les séquences acceptées font partie du langage désiré. Cependant, les séquences $(^n)^n$ pour $n > 4$ ne sont pas acceptées. Il y a une infinité de séquences rejetées qui devraient être acceptées. Pour accepter cette infinité de séquences, il faut ajouter des boucles. Faisons la tentative suivante.



Ce nouvel automate rejette $({}^{5})^{5}, ({}^{9})^{9}$ et des tas d'autres séquences qu'il devrait accepter, mais au moins, il accepte toutes les séquences de la forme $({}^{3m+1})^{3m+1}$ (par exemple $({}^{7})^{7}$). Il accepte donc des séquences de la forme $({}^{n})^{n}$ pour un n arbitrairement grand (c'est-à-dire qu'étant donné n'importe quel entier naturel j, il est possible de trouver un entier $n > j$ tel que $({}^{n})^{n}$ soit acceptée). Malheureusement, cet automate accepte aussi les séquences $({}^{5})^{2}$, $({}^{6})^{3}$ et toutes les séquences de la forme $({}^{3m+1})^{4}$, pour $m > 0$. On voit bien que cela est dû à la boucle d'états 5 3 4 5, qui permet d'ajouter (((autant de fois que désiré avant de passer aux parenthèses fermantes.

Une façon de décrire l'acceptation d'une séquence est de donner la liste des états traversés et, entre chaque paire d'états consécutifs, de donner le symbole lu lors de la transition entre ces deux états. Par exemple, l'acceptation de (()) est décrite par

1 (2 (3) 7) 10.

L'acceptation d'une séquence de la forme $({}^{3m+1})^{4}$ peut donc se visualiser comme suit :

$$1\ \overbrace{(\ 2\ \underbrace{(\ 3\ (\ 4\ (}_{3\text{ fois }(}\ 5\ \underbrace{(\ 3\ (\ 4\ (}_{3\text{ fois }(}\ 5\ \underbrace{(\ 3\ (\ 4\ (}_{3\text{ fois }(}\ 5\ \ldots 4\ (\ 5\ )\ 9\ )\ 12\ )\ 14\ )}^{3m+1\text{ fois }(}\ 15.$$

Pour bien comprendre le théorème qui suit, il faut saisir la signification de « arbitrairement grand ». Un sous-ensemble non vide de $\mathbf{N}$ contient des nombres arbitrairement grands s'il ne contient pas de nombre maximal. C'est donc un ensemble infini, mais qui n'est pas nécessairement tout $\mathbf{N}$. Par exemple, l'ensemble des nombres pairs contient des entiers arbitrairement grands.

1.61 Théorème. *Si un langage régulier L contient des séquences de symboles de la forme $x^n y^n$ pour des $n \in \mathbf{N}$ arbitrairement grands, alors L doit aussi contenir une séquence de la forme $x^m y^n$ où $n \neq m$.*

DÉMONSTRATION. Soit $M = (S, \Sigma, \delta, \iota, F)$ un automate fini déterministe tel que $L(M) = L$ (un tel M existe par la définition 1.57). Il faut montrer que si $L(M)$ contient des séquences de la forme $x^n y^n$ pour des $n \in \mathbf{N}$ arbitrairement grands alors $L(M)$ doit aussi contenir une séquence de la forme $x^m y^n$ où $n \neq m$.

$L(M)$ contient des séquences de la forme $x^n y^n$ pour des n arbitrairement grands

$\Rightarrow$ $\langle$ Signification de *arbitrairement grand.* $\rangle$

$\exists k \in \mathbf{N} : k > |S| \wedge x^k y^k \in L(M)$

$\Rightarrow$ $\langle$ Il y a plus de symboles dans x^k que d'états dans M. Lors de la lecture de x^k, M doit passer au moins deux fois par un certain état, disons s_i. $\rangle$

$\exists k \in \mathbf{N} : \exists s_i \in S : x^k y^k \in L(M) \wedge M$ passe au moins deux fois par s_i en lisant x^k

$\Rightarrow$ $\langle$ Il y a donc une boucle qui permet de partir de s_i et d'y revenir. Supposons que $j > 0$ est la longueur de cette boucle. $\rangle$

$\exists k \in \mathbf{N} : \exists j \in \mathbf{N}^+ : x^k y^k \in L(M) \wedge M$ traverse une boucle de longueur j lors de la lecture de x^k

$\Rightarrow$ $\langle$ Une boucle peut être traversée un nombre quelconque de fois. M peut donc la parcourir une fois de plus. $\rangle$

$\exists k \in \mathbf{N} : \exists j \in \mathbf{N}^+ : x^k y^k \in L(M) \wedge x^{k+j} y^k \in L(M)$

$\Rightarrow$ $\langle$ D'où le résultat annoncé. $\rangle$

$L(M)$ contient une séquence de la forme $x^m y^n$ avec $n \neq m$.

Pour mieux suivre la démonstration ci-dessus, voici deux schémas, le premier décrivant l'acceptation de $x^k y^k$ et le second l'acceptation de $x^{k+j} y^k$. $s_0, s_1, \ldots$ sont des états de M.

$$s_0 \overbrace{x\ s_1\ x\ \ldots s_i\ \underbrace{x\ s_{i+1}\ \ldots x}_{j \text{ fois } x}\ s_i\ \ldots x}^{k \text{ fois } x}\ s_p\ \overbrace{y\ s_{p+1}\ \ldots y}^{k \text{ fois } y}\ s_q.$$

$$s_0 \overbrace{x\ s_1\ x\ \ldots s_i\ \underbrace{x\ s_{i+1}\ \ldots x}_{j \text{ fois } x}\ s_i\ \underbrace{x\ s_{i+1}\ \ldots x}_{j \text{ fois } x}\ s_i\ \ldots x}^{k+j \text{ fois } x}\ s_p\ \overbrace{y\ s_{p+1}\ \ldots y}^{k \text{ fois } y}\ s_q.$$

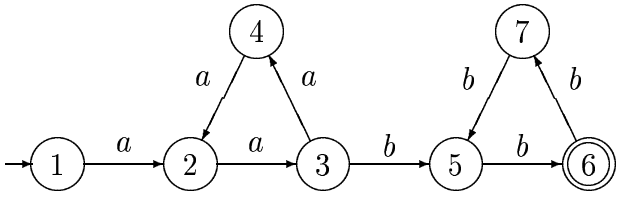
□

Remarquons que le théorème démontre que $x^{k+j}y^k$ est acceptée, mais qu'en fait

$$x^{k-j}y^k, x^{k+2j}y^k, x^{k+3j}y^k, x^{k+4j}y^k, \ldots$$

doivent aussi être acceptées.

1.62 Exemple. La discussion qui précède le théorème 1.61 a illustré la difficulté de reconnaître $(^n)^n$ sans aussi accepter des séquences qui ne sont pas de cette forme. En voici une autre illustration. L'automate fini non ambigu suivant (qui pourrait être complété en un automate déterministe) accepte les séquences de l'ensemble $\{a^{3n+2}b^{3n+2} : n \in \mathbf{N}\}$, par exemple *aabb* et *aaaaabbbbb*, mais il accepte aussi les séquences de la forme $a^{3m+2}b^{3n+2}$ où $m, n \in \mathbf{N}$ et $m \neq n$ (d'où $3m+2 \neq 3n+2$). Ainsi, *aaaaabb* est acceptée.



□

1.63 Exercice. Montrez qu'il n'existe aucun automate fini déterministe M qui a pour langage $L(M) = \{a^n b^n : n \in \mathbf{N}\}$.

□

Pour prouver qu'un langage L n'est pas régulier, une preuve par contradiction est souvent appropriée. Une méthode qui est parfois utile est la suivante : On suppose qu'il existe un automate fini M tel que $L(M) = L$. On choisit ensuite une séquence $w \in L$ suffisamment longue (plus longue que le nombre d'états de M). La reconnaissance de w ne peut donc se faire sans qu'il y ait une boucle (c'est l'idée utilisée dans le théorème 1.61). On montre ensuite qu'à cause de cette boucle, M doit reconnaître une séquence $v \notin L$, d'où $L(M) \neq L$. C'est une contradiction.

1.64 Exemple. Montrez que le langage $L = \{a^{n^2} : n \in \mathbf{N}\}$ n'est pas régulier.

L régulier

$\Rightarrow$ $\langle$ Définition 1.57. $\rangle$

$\exists M : M = (S, \Sigma, \delta, \iota, F) \wedge L(M) = L$

$\Rightarrow$ $\langle$ Parce que L est infini, il contient des séquences arbitrairement longues. $\rangle$

$\exists M : \exists k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \wedge L(M) = L \wedge k > |S| \wedge a^{k^2} \in L(M)$

$\Rightarrow$ $\langle$ Puisqu'il y a plus de $|S|$ symboles a à lire, il doit y avoir une boucle de longueur $j > 0$ et $j \leq |S|$. Cette boucle peut être traversée une fois de plus. $\rangle$

$\exists M : \exists j, k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \wedge L(M) = L \wedge k > |S| \wedge a^{k^2} \in L(M)$
$\wedge\ 0 < j \leq |S| \wedge a^{k^2+j} \in L(M)$

$\Rightarrow$ $\langle$ $j \leq |S| < k \Rightarrow k^2 < k^2 + j < k^2 + k < (k+1)^2$. Par conséquent, $k^2 + j$ ne peut être un carré, d'où $a^{k^2+j} \notin L$. $\rangle$

$\exists M : \exists j, k \in \mathbf{N} : L(M) = L \wedge a^{k^2+j} \in L(M) \wedge a^{k^2+j} \notin L$

$\Rightarrow$ $\langle$ $a^{k^2+j} \in L(M) \wedge a^{k^2+j} \notin L \Rightarrow L(M) \neq L$ $\rangle$

$\exists M : L(M) = L \wedge L(M) \neq L$

$\Leftrightarrow$ $\langle$ $(p \wedge \neg p) \Leftrightarrow$ faux et $(\exists x : \text{faux}) \Leftrightarrow$ faux. $\rangle$

faux.

□

1.65 Exercice. Montrez que les langages suivants ne sont pas réguliers.

1. $\{a^{n!} : n \in \mathbf{N}\}$
2. $\{a^n : n \in \mathbf{N} \wedge n$ n'est pas un carré$\}$ (Indice : regardez l'exercice 1.58 et l'exemple 1.64.)

□

En résumé, les automates finis permettent de reconnaître les mots réservés, les identificateurs et les nombres d'un programme, mais pas les expressions arithmétiques parenthésées ni les structures de blocs `begin end`. Ils ne suffisent donc pas à l'analyse des textes informatiques. Dans le prochain chapitre, nous définirons des machines plus puissantes qui permettront de reconnaître une plus grande gamme de langages.

1.3.3 Problèmes

1. Montrez que l'ensemble des séquences sur l'alphabet $\{a, b, c\}$ contenant un nombre impair de a, un nombre impair de b et un nombre pair de c est un langage régulier.
2. Montrez la régularité du langage sur $\{a, b\}$ dont les éléments sont exactement les séquences qui ne contiennent pas trois a consécutifs.
3. Montrez que le langage $\{a^n : n \text{ est premier}\}$ n'est pas régulier.
4. Montrez que le langage $L = \{a^n b^n c^n : n \in \mathbf{N}\}$ n'est pas régulier.
5. Montrez que le langage L ayant pour alphabet $\{a, b, c\}$ et contenant exactement tous les palindromes n'est pas régulier. Rappelons qu'un palindrome est une séquence de symboles $x_1 x_2 \ldots x_n$ telle que $x_1 x_2 \ldots x_n = x_n x_{n-1} \ldots x_1$. Par exemple, *abba* et *abcba* sont des palindromes.

   Par contre, si L est un langage régulier, est-ce que la collection des palindromes de L est un langage régulier ? Pourquoi ?
6. Soit L le langage avec alphabet $\Sigma = \{a, b\}$ dont les séquences ont le même nombre de a que de b. Montrez que L n'est pas régulier.

1.4 Automates finis non déterministes

OBJECTIFS SPÉCIFIQUES

1. Classifier des automates finis décrits par des diagrammes de transitions. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix.
2. Classifier des descriptions formelles d'automates finis. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix.
3. Démontrer le fonctionnement d'un automate fini donné en énumérant les configurations du ruban et les états consécutifs à une configuration initiale donnée.
4. Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate fini donné.
5. Transformer un automate non déterministe en automate déterministe.
6. Décrire le langage accepté par un automate fini donné.
7. Construire un automate fini acceptant un langage donné.

Par les différents exercices de la section précédente, nous voyons à quel point les automates finis déterministes nous limitent dans l'évaluation syntaxique. Nous devons donc essayer d'élargir la classe de langages que l'on peut analyser. Il serait intéressant de garder le même formalisme que nous avons développé en introduisant les diagrammes de transitions et les automates déterministes.

Comme première tentative, il est naturel de lever deux restrictions que nous avons imposées aux automates déterministes : celle d'être complètement définis et celle d'être non ambigus. Nous obtenons ainsi des automates finis *non déterministes*. Cet ensemble d'automates déterminera une classe de langages qui englobera la classe de langages des automates finis déterministes. Pour tenir compte des différences avec les automates déterministes, nous devrons modifier la définition d'acceptation. Nous nous demanderons ensuite si les automates finis non déterministes permettent de reconnaître des langages non reconnaissables par les automates déterministes.

1.66 Définition. Un *automate fini non déterministe* consiste en un quintuple de la forme $(S, \Sigma, \rho, \iota, F)$. Voici la description des différentes composantes.

- S est un ensemble fini d'états.
- Σ est l'alphabet de la machine.
- ρ est un sous-ensemble de $S \times \Sigma \times S$.

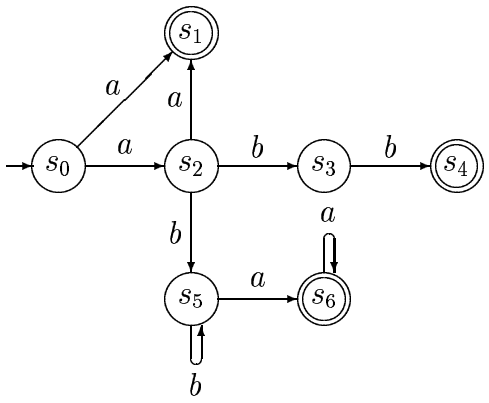
- $\iota \in S$ est l'état initial.
- $F \subseteq S$ est l'ensemble des états finaux.

□

La seule composante d'un automate fini non déterministe différente de celle de son homologue déterministe est la troisième, ρ, qui décrit l'ensemble des transitions de l'automate. Un élément de ρ est un triplet de la forme (s, a, t) où t est un état accessible à partir de l'état s à la lecture d'un a. Remarquez qu'on peut écrire ce triplet comme une paire dont la première composante est elle-même une paire : $((s, a), t)$. Sous cette forme, on voit bien que ρ est une relation entre $S \times \Sigma$ et S, c'est-à-dire que $\rho \subseteq (S \times \Sigma) \times S$. Ce qui distingue les automates non déterministes des automates déterministes est donc la définition des transitions au moyen d'une relation plutôt que d'une fonction.

1.67 Exemple. Voici le diagramme de transitions d'un automate non déterministe

$$(\{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}, \{a, b\}, \rho, s_0, \{s_1, s_4, s_6\}).$$



La relation de transition est

$$\rho = \{(s_0, a, s_1), (s_0, a, s_2), (s_2, a, s_1), (s_2, b, s_3), (s_2, b, s_5), \\ (s_3, b, s_4), (s_5, a, s_6), (s_5, b, s_5), (s_6, a, s_6)\}.$$

Cet automate n'est pas déterministe, car

- il n'est pas complètement défini : il n'y a pas de transition pour b à partir de l'état s_0.
- il est ambigu : il y a deux transitions pour le symbole a à partir de l'état s_0.

Notez qu'une seule des deux raisons suffit. □

1.68 Exercice. Donnez le diagramme de transitions de l'automate

$$(\{A, B, C, D, E, F, G, H, I\}, \{0, 1\}, \rho, A, \{G, I\}),$$

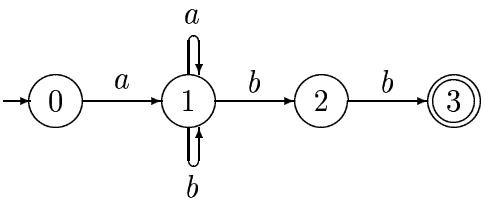
où

$$\rho = \{(A,1,B),(A,1,C),(B,0,E),(C,0,B),(C,0,D),(C,1,F)$$
$$(D,0,A),(D,0,G),(D,1,C),(D,1,D),(E,0,H),(E,1,I),$$
$$(F,1,G),(F,1,I),(G,0,F),(G,1,G),(H,0,H),(I,1,H)\}.$$

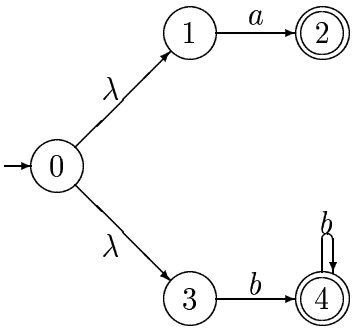
□

1.69 Exercice. Classifiez les automates finis ci-dessous décrits par des diagrammes de transitions sur l'alphabet Σ. Dites tout d'abord si la description est correcte. Si oui, dites si l'automate est déterministe. Expliquez la raison de chaque choix.

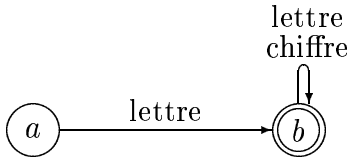
1. $\Sigma = \{a, b\}$



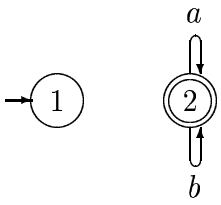
2. $\Sigma = \{a, b\}$



3. $\Sigma =$ lettre $\cup$ chiffre

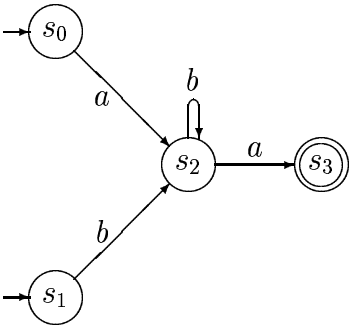


4. $\Sigma = \{a, b\}$

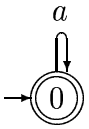


5. $\Sigma = \emptyset$

6. $\Sigma = \{a, b\}$

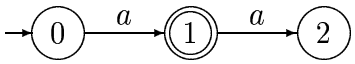


7. $\Sigma = \{a\}$



8. $\Sigma = \{a\}$

9. $\Sigma = \{a\}$



□

1.70 Exercice. Classifiez les descriptions formelles suivantes d'automates finis. Dites tout d'abord si la description est correcte. Si oui, dites si l'automate est déterministe. Expliquez la raison de chaque choix.

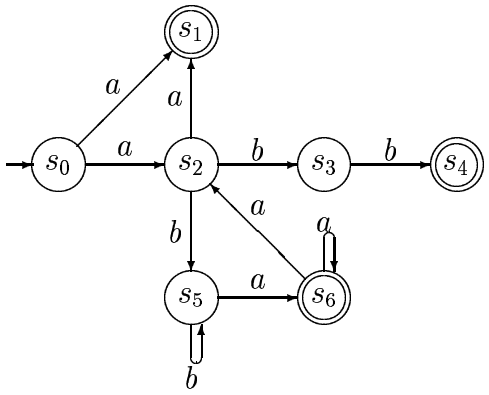
1. $M_1 = (\{s_0, s_1, s_2\}, \{a, b\}, \delta, s_0, \{s_1\})$ où
$$\delta(s_0, a) = s_1 \quad \delta(s_0, b) = s_2$$
$$\delta(s_1, a) = s_1 \quad \delta(s_1, b) = s_2$$
$$\delta(s_2, a) = s_2 \quad \delta(s_2, b) = s_2$$
2. $M_2 = (\{0\}, \Sigma, \delta, \{0\})$ où
$$\Sigma = \{a, b, c\}$$
$$\delta(0, a) = 0 \quad \delta(0, b) = 0 \quad \delta(0, c) = 0$$
3. $M_3 = (\{a, b\}, \{1, 2\}, \rho, a, \emptyset)$ où
$$\rho(a, 1) = a \quad \rho(a, 2) = b$$
$$\rho(b, 1) = a \quad \rho(b, 2) = b$$
4. $M_4 = (\{s_1, s_2, s_3, s_4\}, \{a, b\}, \rho, s_1, \{s_2, s_4\})$ où
$$\rho = \{(s_1, a, s_2), (s_1, b, s_2)$$
$$(s_2, a, s_3), (s_2, b, s_4)$$
$$(s_3, a, s_4), (s_3, b, s_2)\}$$
5. $M_5 = (\{1, 2, 3, 4\}, \{a, b, c\}, \delta, 1, \{2\})$ où
$$\delta = \{(1, a, 2), (1, b, 3), (1, c, 3)$$
$$(2, a, 2), (2, b, 4), (2, c, 4)$$
$$(2, a, 3), (2, b, 3), (2, c, 3)$$
$$(3, a, 3), (3, b, 3), (3, c, 3)$$
$$(4, a, 3), (4, b, 3), (4, c, 3)\}$$
6. $M_6 = (\{a, b\}, \{1\}, \delta = \emptyset, a, \{b\})$

□

1.71 Définition. Un automate fini non déterministe $M = (S, \Sigma, \rho, \iota, F)$ *accepte* (ou *reconnaît*) la séquence $x_1x_2x_3 \ldots x_n (n \geq 0)$ si, et seulement si, il *existe* une séquence d'états $s_0, s_1, \ldots, s_n$ tels que $s_0 = \iota, s_n \in F$, et pour $j = 1, \ldots, n$, $(s_{j-1}, x_j, s_j) \in \rho$. Dans le cas contraire, on dit que l'automate refuse la séquence. On dénote par $L(M)$ l'ensemble des séquences acceptées par M. On dit que $L(M)$ est le *langage reconnu* (ou *accepté*) par M. □

Il est important de noter qu'une séquence est acceptée *s'il est possible* d'atteindre un état final en lisant toute la séquence ; même s'il y a en plus des chemins qui mènent à des états non finaux, la séquence est acceptée.

1.72 Exemple. Soit l'automate non déterministe M décrit par le diagramme de transitions suivant :



Voici une suite de configurations du ruban et d'ensembles d'états de l'automate décrivant l'acceptation de la séquence *abbbaa* par M. Les ensembles d'états contiennent tous les états accessibles à une étape donnée.

$\underline{a}bbbaa$	$\{s_0\}$
$a\underline{b}bbaa$	$\{s_1, s_2\}$
$ab\underline{b}baa$	$\{s_3, s_5\}$
$abb\underline{b}aa$	$\{s_4, s_5\}$
$abbb\underline{a}a$	$\{s_5\}$
$abbba\underline{a}$	$\{s_6\}$
$abbbaa\_$	$\{s_2, s_6\}$

On pourrait aussi décrire l'acceptation de la séquence *abbbaa* en généralisant la méthode de description utilisée dans la section précédente. La description appropriée du chemin suivi est, selon la liste de configurations ci-dessus,

$$\{s_0\}\ a\ \{s_1, s_2\}\ b\ \{s_3, s_5\}\ b\ \{s_4, s_5\}\ b\ \{s_5\}\ a\ \{s_6\}\ a\ \{s_2, s_6\}.$$

La séquence est acceptée, car il existe une suite de transitions (correspondant à la séquence) menant de l'état initial s_0 à l'état final s_6. Une telle suite de transitions est

$$s_0\ a\ s_2\ b\ s_5\ b\ s_5\ b\ s_5\ a\ s_6\ a\ s_6.$$

L'état s_6 est bien un élément de l'ensemble des états possibles après la lecture de la séquence (dernier ensemble de la suite ci-dessus). Même si l'automate peut atteindre l'état non final s_2, la séquence est quand même acceptée.

Remarquez comment une transition est décrite comme le passage d'un ensemble d'états vers un autre ensemble d'états.

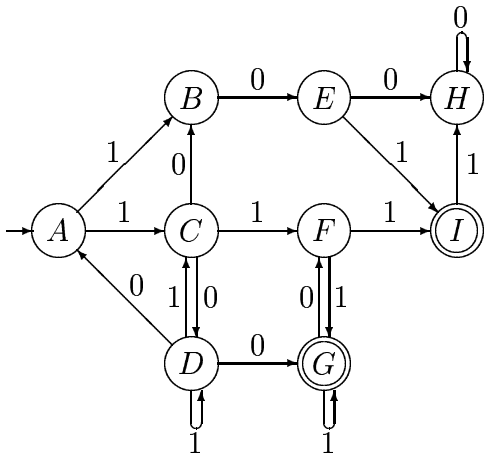
La suite ci-dessous décrit le rejet de la séquence *abbbb* par le même automate. Celui-ci la rejette car il n'y a pas de suite de transitions correspondant à la séquence

et menant de l'état initial à un état final ; cela se déduit facilement du fait qu'il n'y a pas d'état final dans l'ensemble d'états possibles après la lecture de la séquence.

$\underline{a}bbbb$	$\{s_0\}$
$a\underline{b}bbb$	$\{s_1, s_2\}$
$ab\underline{b}bb$	$\{s_3, s_5\}$
$abb\underline{b}b$	$\{s_4, s_5\}$
$abbb\underline{b}$	$\{s_5\}$
$abbbb\_$	$\{s_5\}$

□

1.73 Exercice. Soit un automate M' décrit par le diagramme de transitions suivant :



Démontrer le fonctionnement de M' (en énumérant les configurations du ruban d'entrée et les ensembles d'états possibles de l'automate) lorsqu'on lui soumet les séquences suivantes. Dites, dans chaque cas, si la chaîne est acceptée ou rejetée.

1. 10000
2. 10011
3. 1101

□

L'exemple 1.72 et l'exercice 1.73 montrent comment on peut analyser l'acceptation des séquences par un automate fini non déterministe en considérant des transitions entre des sous-ensembles de l'ensemble des états de l'automate. Le théorème suivant utilise cette méthode. Référez donc à l'exemple et à l'exercice sus-mentionnés pour mieux comprendre ce théorème.

1.74 Théorème. *Pour tout automate fini non déterministe, il existe un automate fini déterministe qui accepte exactement le même langage.*

DÉMONSTRATION. Soit un automate fini non déterministe

$$M = (S, \Sigma, \rho, \iota, F).$$

Nous allons démontrer l'existence d'un automate fini déterministe

$$M' = (S', \Sigma, \delta, \iota', F')$$

tel que $L(M') = L(M)$ (remarquons que l'alphabet ne change pas). L'idée de la preuve est de suivre toutes les options possibles en parallèle ; chaque transition passe d'un sous-ensemble de S à un sous-ensemble de S.

Posons

$$S' = \wp(S)$$
$$\iota' = \{\iota\}$$
$$F' = \{T \subseteq S : T \cap F \neq \emptyset\}.$$

En mots : un état de M' est un sous-ensemble d'états de M ; l'état initial est l'ensemble qui contient l'état initial de M ; un état de M' (un sous-ensemble de S) est final ssi il contient au moins un état final de M.

La fonction $\delta : S' \times \Sigma \to S'$ est définie par

$$\delta(s', x) = \{s \in S : \exists r \in s' : (r, x, s) \in \rho\}.$$

Autrement dit, à la lecture d'un x, M' passe de l'état s' à l'ensemble des états accessibles par M à partir des états dans s'. Notez que δ est bien une fonction. Elle est définie pour tout $s' \in \wp(S)$ et tout $x \in \Sigma$. De plus, le résultat est unique (non ambigu).

Maintenant que M' est défini, il nous faut montrer que $L(M) = L(M')$. Procédons en deux étapes : la première montrera que $L(M) \subseteq L(M')$ et la deuxième que $L(M) \supseteq L(M')$.

1. $L(M) \subseteq L(M')$:

   Montrons par induction sur n que le prédicat $P(n)$ suivant est vrai pour tout $n \in \mathbf{N}$:

   Si

   $$s_0 = \iota \quad x_1 \quad s_1 \quad x_2 \quad s_2 \quad \ldots \quad x_n \quad s_n$$

   est un chemin de M, alors il existe un chemin de M' de la forme

   $$s'_0 = \iota' \quad x_1 \quad s'_1 \quad x_2 \quad s'_2 \quad \ldots \quad x_n \quad s'_n$$

   (i.e. faisant la lecture des mêmes symboles que M) et tel que $s_n \in s'_n$.

   De manière abrégée, nous écrirons $P(n)$ de la façon suivante :

   $$\begin{array}{rlllllll} & s_0 = \iota & x_1 & s_1 & \ldots & x_n & s_n & \text{de } M \\ \Rightarrow \exists & s'_0 = \iota' & x_1 & s'_1 & \ldots & x_n & s'_n & \text{de } M' \wedge s_n \in s'_n. \end{array}$$

- Base d'induction : $n = 0$. Le chemin de M est de longueur 0 (M reste dans l'état initial $s_0 = \iota$). Il existe un chemin correspondant pour M' : le chemin de longueur 0 ; M' reste dans l'état initial $s'_0 = \iota'$. On a bien $s_0 \in s'_0$, par définition de ι'.
- Étape d'induction. Montrons que $P(k) \Rightarrow P(k+1)$. Considérons que $P(k)$ est vrai (hypothèse d'induction), c'est-à-dire que

$$\begin{array}{ll} & s_0 = \iota \quad x_1 \quad s_1 \quad \ldots \quad x_k \quad s_k \quad \text{de } M \\ \Rightarrow \exists & s'_0 = \iota' \quad x_1 \quad s'_1 \quad \ldots \quad x_k \quad s'_k \quad \text{de } M' \wedge s_k \in s'_k. \end{array}$$

Nous allons démontrer $P(k+1)$ en faisant appel à la vérité de $P(k)$ au moment opportun, montrant ainsi que $P(k)$ implique $P(k+1)$.

$$\begin{array}{ll} & s_0 = \iota \quad x_1 \quad s_1 \quad \ldots \quad x_k \quad s_k \quad x_{k+1} \quad s_{k+1} \quad \text{de } M \\ \Leftrightarrow & \quad \langle \text{ Signification de } \textit{chemin}. \rangle \\ & s_0 = \iota \quad x_1 \quad s_1 \quad \ldots \quad x_k \quad s_k \quad \text{de } M \wedge (s_k, x_{k+1}, s_{k+1}) \in \rho \\ \Rightarrow & \quad \langle \text{ Par l'hypothèse d'induction. } \rangle \\ & \exists \quad s'_0 = \iota' \quad x_1 \quad s'_1 \quad \ldots \quad x_k \quad s'_k \quad \text{de } M' \\ & \wedge\ s_k \in s'_k \wedge (s_k, x_{k+1}, s_{k+1}) \in \rho \\ \Rightarrow & \quad \langle \text{ Définition de } \delta. \rangle \\ & \exists \quad s'_0 = \iota' \quad x_1 \quad s'_1 \quad \ldots \quad x_k \quad s'_k \quad \text{de } M' \wedge s_{k+1} \in \delta(s'_k, x_{k+1}) \\ \Rightarrow & \quad \langle \text{ Posons } s'_{k+1} = \delta(s'_k, x_{k+1}). \rangle \\ & \exists \quad s'_0 = \iota' \quad x_1 \quad s'_1 \quad \ldots \quad x_k \quad s'_k \quad x_{k+1} \quad s'_{k+1} \quad \text{de } M' \\ & \wedge\ s_{k+1} \in s'_{k+1}. \end{array}$$

Par conséquent,

$$\begin{array}{ll} & M \text{ accepte } x_1 x_2 \ldots x_n \\ \Leftrightarrow & \quad \langle \text{ Définition d'acceptation pour un automate non déterministe. } \rangle \\ & \exists \text{ un état final } s_n \text{ et un chemin } s_0 = \iota \quad x_1 \quad s_1 \quad \ldots \quad x_n \quad s_n \quad \text{de } M \\ \Rightarrow & \quad \langle \text{ Preuve ci-dessus et définition de } F'. \rangle \\ & \exists \text{ un état final } s'_n \text{ et un chemin } s'_0 = \iota' \quad x_1 \quad s'_1 \quad \ldots \quad x_n \quad s'_n \quad \text{de } M' \\ \Rightarrow & \quad \langle \text{ Définition d'acceptation pour un automate déterministe. } \rangle \\ & M' \text{ accepte } x_1 x_2 \ldots x_n. \end{array}$$

Puisque toute séquence acceptée par M est acceptée par M', $L(M) \subseteq L(M')$.

2. $L(M) \supseteq L(M')$:
Montrons par induction sur n que

pour chaque chemin de l'automate M' partant de l'état initial ι' jusqu'à l'état $s'_n \in F'$ avec $s_n \in s'_n$, il existe un chemin dans l'automate M partant de l'état ι et se rendant jusqu'à l'état final $s_n \in F$ en traversant les arcs étiquetés $x_1, x_2, \ldots, x_n$.

- Base d'induction : $n = 0$. Comme pour le cas précédent.
- Étape d'induction : Supposons que le résultat est vrai pour k (hypothèse d'induction) et considérons un chemin dans M' de ι' jusqu'à s'_{k+1} traversant les arcs $x_1, \ldots, x_k$ et x_{k+1}. Par l'hypothèse d'induction, pour chaque $s_k \in s'_k$, il doit exister un chemin dans M de ι jusqu'à s_k traversant les arcs étiquetés $x_1, x_2, \ldots, x_k$. Mais
$$\delta(s'_k, x_{k+1}) = s'_{k+1} = \{s \in S | (s_k, x_{k+1}, s) \in \rho\}.$$
Alors, pour chaque $s \in s'_{k+1}$, il existe un chemin dans M de ι jusqu'à s traversant les arcs étiquetés $x_1, x_2, \ldots, x_k$ et x_{k+1} comme demandé.

□

1.75 Remarque. Donc, on ne peut augmenter la puissance d'un automate fini en lui permettant d'être non déterministe, c'est-à-dire que les langages reconnus par les automates finis non déterministes sont les mêmes que les langages reconnus par les automates finis déterministes. Notez que le nombre d'états augmente exponentiellement lorsqu'on transforme un automate non déterministe en un automate déterministe. Un programme simulant un automate non déterministe doit

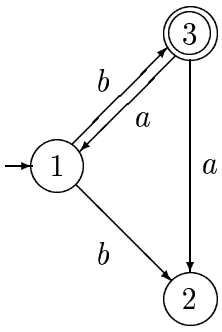
- ou bien explorer les chemins possibles un à un (avec retour arrière) : cela peut être long.
- ou bien construire l'automate déterministe correspondant : cela peut nécessiter beaucoup d'espace.

On peut dire qu'un langage régulier est un langage reconnu par un automate fini, déterministe ou non. □

1.76 Exemple. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisons un automate fini déterministe qui reconnaît les mêmes séquences que l'automate non déterministe $M = (S, \Sigma, \rho, \iota, F)$ où

- $S = \{1, 2, 3\}$
- $\Sigma = \{a, b\}$
- $\rho = \{(1, b, 2), (1, b, 3), (3, a, 1), (3, a, 2)\}$
- $\iota = 1$.
- $F = \{3\}$

Le diagramme de transitions est le suivant.

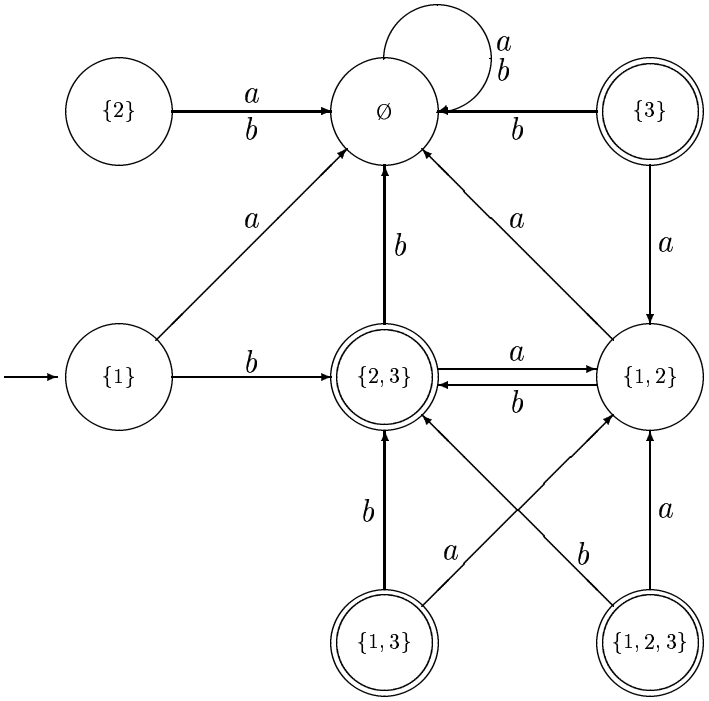


La construction de l'automate fini déterministe se fait à partir des résultats suivants :

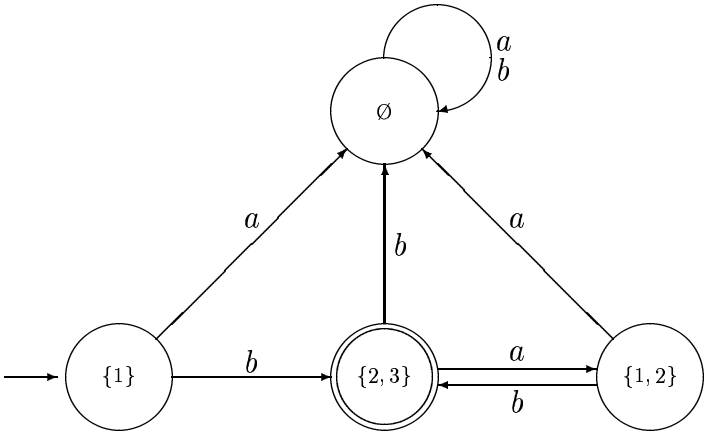
- $S' = \wp(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.
- Σ ne change pas.
- δ est la fonction suivante :

$\delta(\emptyset, a) = \emptyset$	$\delta(\emptyset, b) = \emptyset$
$\delta(\{1\}, a) = \emptyset$	$\delta(\{1\}, b) = \{2, 3\}$
$\delta(\{2\}, a) = \emptyset$	$\delta(\{2\}, b) = \emptyset$
$\delta(\{3\}, a) = \{1, 2\}$	$\delta(\{3\}, b) = \emptyset$
$\delta(\{1, 2\}, a) = \emptyset$	$\delta(\{1, 2\}, b) = \{2, 3\}$
$\delta(\{1, 3\}, a) = \{1, 2\}$	$\delta(\{1, 3\}, b) = \{2, 3\}$
$\delta(\{2, 3\}, a) = \{1, 2\}$	$\delta(\{2, 3\}, b) = \emptyset$
$\delta(\{1, 2, 3\}, a) = \{1, 2\}$	$\delta(\{1, 2, 3\}, b) = \{2, 3\}$

- $\iota' = \{\iota\} = \{1\}$
- $F' = \{\{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$



Si on supprime les états qui ne peuvent être atteints, on obtient l'automate fini déterministe suivant.

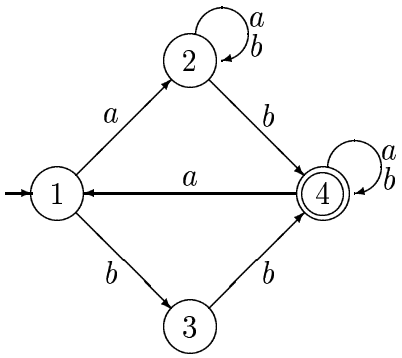


Il est ainsi plus facile de voir que le langage que l'automate accepte est

$$\{b(ab)^n : n \in \mathbf{N}\}.$$

□

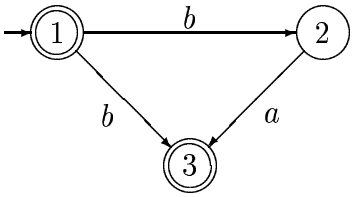
1.77 Exercice. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un automate fini déterministe qui accepte les mêmes séquences que l'automate fini non déterministe ci-dessous. L'alphabet est $\{a, b\}$.



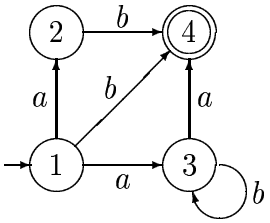
□

1.4.1 Problèmes

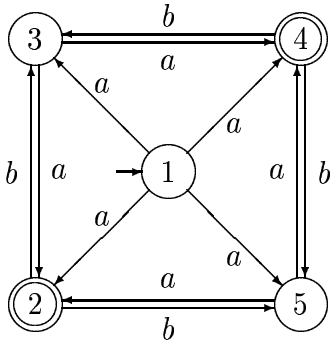
1. Montrez la régularité du langage sur $\{a, b\}$ dont les éléments sont exactement les séquences qui contiennent un nombre pair de a et un nombre pair de b ou un nombre de a divisible par 3 et un nombre de b divisible par 3.
2. Montrez que chaque automate fini déterministe $(S, \Sigma, \delta, \iota, F)$ est aussi un automate fini non déterministe.
3. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un automate fini déterministe qui accepte les mêmes séquences que l'automate fini non déterministe ci-dessous. L'alphabet est $\{a, b\}$.



4. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un diagramme de transitions pour un automate fini déterministe qui accepte le même langage que l'automate fini non déterministe représenté ci-dessous. L'alphabet est $\{a, b\}$.



5. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un automate fini déterministe qui reconnaît les mêmes séquences que l'automate non déterministe ci-dessous. L'alphabet est $\{a, b\}$.



1.5 Grammaires régulières

OBJECTIFS SPÉCIFIQUES

1. Décrire le langage généré par une grammaire régulière donnée.
2. Construire une grammaire régulière générant un langage donné.
3. Les grammaires régulières et les automates finis décrivent les mêmes langages : Passer d'une représentation à l'autre.

Dans cette section, nous introduisons un nouvel outil servant à la génération de langages : les grammaires. Remarquez bien que cet outil ne servira pas à la reconnaissance des séquences d'un langage, comme c'était le cas pour les automates, mais à leur génération. Vous êtes déjà familiers avec les grammaires BNF. Le principe des grammaires présentées ici est très semblable. Voici la définition d'une grammaire.

1.78 Définition. Une *grammaire* (ou encore, *grammaire de phrases* ou *grammaire à structures de phrase*) est un quadruplet (V, T, S, R) où

- V est un ensemble fini de *symboles non terminaux* (alphabet non terminal).
- T est un ensemble fini de *symboles terminaux* (alphabet terminal).
- $S \in V$ est le *symbole de départ* qu'on nomme aussi *axiome* ou *symbole initial*.
- R est un ensemble fini de *règles de réécriture* (ou *productions* ou *règles de substitution*). Ces règles sont formées d'un terme de gauche, d'une flèche ($\rightarrow$) et d'un terme de droite. Les termes gauche et droit peuvent être n'importe quelle combinaison de symboles de V ou T, pourvu qu'il y ait au moins un symbole de V à gauche. Le côté droit peut être vide, ce qui est indiqué par un λ. On parle alors de *règle-λ*.

□

1.79 Exemple. Voici une grammaire de phrases (V, T, S, R) où

- $V = \{S, A, B, C\}$;
- $T = \{a, b, c\}$;
- S est le symbole de départ ;
- $R = \{S \rightarrow ASC$
  $S \rightarrow B$
  $B \rightarrow bB$
  $B \rightarrow \lambda$
  $A \rightarrow a$
  $C \rightarrow c\}$

  La règle $B \rightarrow \lambda$ est une règle-λ.

□

1.80 Remarque. À moins d'avis contraire, nous appliquerons la convention suivante pour la construction des grammaires :

- Nous utiliserons des majuscules pour les symboles non terminaux.
- Nous utiliserons des minuscules pour les symboles terminaux.
- Le symbole initial sera S.

Cette convention permet de simplifier la description d'une grammaire en donnant seulement la description de R. □

Une grammaire génère une séquence de symboles terminaux par dérivation. Une dérivation est une suite d'étapes où chaque étape consiste à remplacer (tant que c'est possible) un patron apparaissant du côté gauche d'une règle de R par le côté droit correspondant. La chaîne initiale est le symbole de départ de la grammaire.

1.81 Exemple. À partir de l'exemple 1.79, on obtient la dérivation suivante :

$$
\begin{aligned}
& S \\
\Rightarrow \quad & \langle \text{ Production } S \to ASC. \rangle \\
& ASC \\
\Rightarrow \quad & \langle \text{ Production } A \to a. \rangle \\
& aSC \\
\Rightarrow \quad & \langle \text{ Production } S \to B. \rangle \\
& aBC \\
\Rightarrow \quad & \langle \text{ Production } B \to bB. \rangle \\
& abBC \\
\Rightarrow \quad & \langle \text{ Production } B \to \lambda. \rangle \\
& abC \\
\Rightarrow \quad & \langle \text{ Production } C \to c. \rangle \\
& abc
\end{aligned}
$$

Parce que la règle de réécriture utilisée à chaque étape est en général évidente, nous écrirons les dérivations sous une forme plus compacte. Par exemple, la dérivation ci-dessus s'écrit

$$S \Rightarrow ASC \Rightarrow aSC \Rightarrow aBC \Rightarrow abBC \Rightarrow abC \Rightarrow abc$$

La transformation résultant de l'utilisation d'une règle de réécriture est indiquée par la flèche $\Rightarrow$ (c'est une erreur d'utiliser la flèche $\to$ dans une dérivation ; elle sert à

définir les productions). Il n'y pas d'ordre d'utilisation des règles de dérivation. On peut aussi obtenir la séquence abc avec la dérivation suivante :

$$S \Rightarrow ASC \Rightarrow ASc \Rightarrow aSc \Rightarrow aBc \Rightarrow abBc \Rightarrow abc$$

□

Voici maintenant un exemple d'une grammaire où le côté gauche des productions est plus complexe.

1.82 Exemple. Les règles de réécriture sont

$$\begin{array}{ll} S \to abNSc & bNa \to abN \\ S \to \lambda & bNb \to bbN \\ & bNc \to bc \end{array}$$

On peut dériver la séquence $aabbcc$ comme suit :

$$\begin{aligned} S &\Rightarrow abNSc \Rightarrow abNabNScc \Rightarrow abNabNcc \\ &\Rightarrow aabNbNcc \Rightarrow aabNbcc \Rightarrow aabbNcc \Rightarrow aabbcc \end{aligned}$$

□

1.83 Exercice. Soit $G = (V, T, S, R)$ la grammaire dont les règles de réécriture sont :

$$\begin{array}{lll} S \to AB & A \to aA & B \to Bb \\ & A \to \lambda & B \to b \end{array}$$

1. Sachant que la présentation de cette grammaire suit la convention décrite à la remarque 1.80, donnez V et T.
   - $V =$
   - $T =$
2. Dites si les séquences suivantes sont générées par cette grammaire. Si oui, donnez une dérivation ; si non, dites pourquoi.
   - $aabb$
   - $abbb$
   - λ
   - ba

□

1.84 Définition. Soit $G = (V, T, S, R)$. Si $T \subseteq \Sigma$ où Σ est un alphabet, on dit que G est une grammaire sur Σ. Soit

$$L(G) = \{w \in T^* : w \text{ est dérivée de } S\}.$$

On dit que $L(G)$ est le *langage généré* par G. □

1.85 Exemple. Reprenons la grammaire G de l'exemple 1.79.

$S \to ASC$	$B \to bB$	$A \to a$
$S \to B$	$B \to \lambda$	$C \to c$

Une application répétée (zéro ou plusieurs fois) de la production $S \to ASC$ conduit à des séquences de la forme A^nSC^n. Éventuellement, la production $S \to B$ est appliquée, donnant A^nBC^n. Par la suite, la règle $B \to bB$ peut être utilisée zéro ou plusieurs fois ; le résultat est une séquence de la forme $A^nb^mBC^n$. L'application de $B \to \lambda$ mène à $A^nb^mC^n$. Finalement, en utilisant n fois $A \to a$ et $C \to c$, nous obtenons $a^nb^mc^n$. Les règles sont suffisamment simples pour voir que tout autre ordre de dérivation conduit à des séquences de la même forme. Le langage généré par G est donc

$$L(G) = \{a^nb^mc^n : m \in \mathbf{N} \wedge n \in \mathbf{N}\}.$$

Ce langage contient λ. En effet, la séquence vide peut être générée par la production $S \to B$ suivie de $B \to \lambda$. □

1.86 Exercice. Dites quel est le langage généré par chacune des grammaires suivantes.

1. | | | |
   |---|---|---|
   | $S \to AB$ | $A \to aA$ | $B \to Bb$ |
   | | $A \to \lambda$ | $B \to b$ |
2. | | |
   |---|---|
   | $S \to aSb$ | $S \to ab$ |

□

1.87 Exercice. Soit la grammaire $(\{\alpha, \beta, \gamma\}, \{a, b\}, \gamma, R)$ où R est l'ensemble des règles suivantes.

$$\begin{array}{ll} \gamma \to a\beta & \alpha \to a\gamma \\ \gamma \to b\alpha & \alpha \to b\gamma \\ \gamma \to \lambda & \end{array}$$

Trouvez une grammaire qui génère le même langage et qui suit la convention décrite à la remarque 1.80.

□

1.88 Définition. Une grammaire *régulière* est une grammaire avec les restrictions suivantes :

- Le côté gauche consiste en un seul symbole non terminal.
- Le côté droit est
  - soit un symbole terminal suivi d'un symbole non terminal,
  - soit un seul symbole terminal,
  - soit λ.

□

1.89 Exemple. La grammaire

$$C \to bB \qquad C \to a \qquad D \to \lambda$$

est une grammaire régulière, mais pas

$$bD \to A \qquad A \to aCb \qquad BA \to DeC$$

□

1.90 Exercice. Les grammaires de l'exercice 1.86 sont-elles régulières ? Expliquez votre réponse.

□

1.91 Exercice. Construisez une grammaire *régulière* générant le langage

$\{a^p bc^q : p \in \mathbf{N} \wedge q \in \mathbf{N}^+\}$.

□

1.92 Remarque. Il est possible d'éliminer les règles dont le côté droit consiste en un seul symbole terminal, c'est-à-dire les règles de la forme $N \to x$, en les remplaçant par

$$N \to xX \qquad X \to \lambda$$

où X est un nouveau symbole n'apparaissant pas dans la grammaire initiale. Évidemment, le langage généré est le même. □

1.93 Exercice.

1. Transformez la grammaire régulière suivante en une autre grammaire régulière générant le même langage, mais ne contenant aucune règle dont le côté droit consiste en un seul symbole terminal.

   $$S \to aS \qquad S \to b \qquad S \to c$$

2. Donnez une dérivation de *aaab* avec les deux grammaires. Remarquez que tout au long de la dérivation, la séquence contient un seul symbole non terminal. Il disparaît lors de l'application de la dernière production.

3. Dites quel est le langage généré par cette grammaire.

□

Nous allons maintenant montrer que les grammaires régulières génèrent exactement les langages reconnus par les automates finis (déterministes ou non déterministes, puisqu'ils acceptent les mêmes langages). Malgré les fortes restrictions imposées à la forme des règles de réécriture des grammaires régulières, elles génèrent l'ensemble des langages reconnaissables par les automates finis. Nous verrons dans les chapitres suivants comment la levée des restrictions imposées aux grammaires régulières permet la génération de nouveaux langages. Mais nous devons auparavant bien comprendre les grammaires régulières et leur équivalence avec les automates finis.

1.94 Théorème. *Pour chaque alphabet* Σ*,*

$$\{L(G) : G \textit{ est une grammaire régulière sur } \Sigma\}$$

$$= \{L(M) : M \textit{ est un automate fini sur } \Sigma\}.$$

(En mots : l'ensemble des langages générés par toutes les grammaires régulières est égal à l'ensemble des langages reconnus par tous les automates finis.)

DÉMONSTRATION. La démonstration indique d'abord comment, à partir d'une grammaire G, on peut construire un automate M tel que $L(G) = L(M)$. Ensuite, on montre comment dériver, à partir d'un automate M, une grammaire G telle que $L(G) = L(M)$. (Puisque ces conversions peuvent toujours être faites, cela démontre le théorème.) Cette technique pourra être utilisée plus tard pour faire des conversions grammaire/automate et automate/grammaire.

- Conversion grammaire/automate.
  Soit $G = (V, T, S_G, R)$ une grammaire régulière.
  1. D'après la remarque 1.92, on peut transformer la grammaire G en une grammaire
     $$G' = (V', T, S_G, R')$$
     qui génère le même langage que G, mais n'ayant pas de règle dont le côté droit consiste en un seul symbole terminal.
  2. On définit ensuite un automate fini non déterministe
     $$M = (S_M, \Sigma, \rho, \iota, F)$$
     comme suit :
     – $S_M = V'$
     – $\Sigma = T$
     – $\rho = \{(P, x, Q) :$ il y a une règle de la forme $P \to xQ$ dans $R'\}$
     – $\iota = S_G$

– $F = \{X : X \to \lambda \text{ est une règle de } R'\}$

De la définition de ρ, on voit qu'il y a lecture d'un x par l'automate ssi il y a génération d'un x par la grammaire. Quant à la définition de F, elle se justifie en remarquant qu'une règle de la forme $X \to \lambda$ fait disparaître le symbole non terminal X. Parce que G' est régulière, il n'y a pas d'autre non terminal à éliminer (voir exemple 1.93). La génération est terminée pour la grammaire ; en rendant X final, l'automate peut accepter la séquence menant à X.

- Conversion automate/grammaire.
  Soit $M = (S_M, \Sigma, \rho, \iota, F)$ un automate fini non déterministe. La grammaire
  $$G = (V, T, S_G, R)$$
  est définie comme suit :
  – $V = S_M$
  – $T = \Sigma$
  – $S_G = \iota$
  – $R = \{P \to xQ : (P, x, Q) \in \rho\} \cup \{Q \to \lambda : Q \in F\}$
  Il s'agit donc tout simplement de la construction inverse.

□

1.95 Exemple. Prenons la grammaire régulière suivante :

$$\begin{array}{lll} S \to aA & A \to bB & B \to aA \\ S \to bB & A \to \lambda & B \to \lambda \end{array}$$

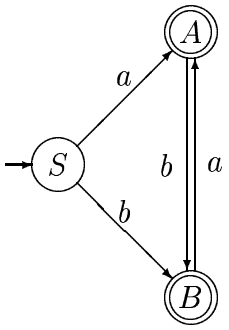
La grammaire n'a pas de règle dont le côté droit consiste en un seul symbole terminal. Il n'est donc pas nécessaire de faire l'étape 1 de la conversion grammaire/automate, décrite dans la preuve du théorème 1.94. Passons à l'étape 2. Puisque

$V = \{S, A, B\}$,
$T = \{a, b\}$,

nous trouvons l'automate $M = (S', \Sigma, \rho, \iota, F)$ suivant.

- $S' = \{A, B, S\}$
- $\Sigma = \{a, b\}$
- $\rho = \{(S, a, A), (S, b, B), (A, b, B), (B, a, A)\}$
- $\iota = S$
- $F = \{A, B\}$

Voici son diagramme de transitions.



On voit bien comment faire la transformation inverse (automate/grammaire) en utilisant la méthode du théorème 1.94. L'automate accepte la séquence aba en passant par les états $SABA$. La grammaire génère cette chaîne par la dérivation

$$S \Rightarrow aA \Rightarrow abB \Rightarrow abaA \Rightarrow aba$$

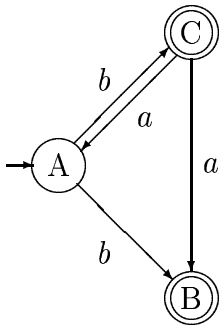
Remarquez que la suite de symboles non terminaux apparaissant dans cette dérivation est $SABA$, c'est-à-dire la même que la suite d'états de l'automate. □

1.96 Exercice. Utilisez la construction du théorème 1.94 pour trouver un automate fini qui reconnaît le même langage que le langage généré par la grammaire régulière suivante.

$S \to aA$ $\quad S \to bS$ $\quad A \to aA$
$S \to a$ $\quad S \to b$ $\quad A \to a$

□

1.97 Exercice. Utilisez la construction du théorème 1.94 pour trouver une grammaire régulière qui génère le même langage que le langage accepté par l'automate fini suivant.



□

1.5.1 Problèmes

1. Construisez une grammaire régulière générant le langage

   $\{ab^m ab^n : m \in \mathbf{N} \wedge n \in \mathbf{N}\}$.

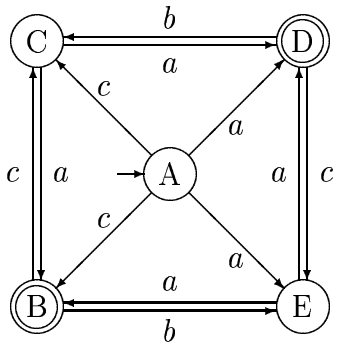
2. Donnez un automate fini qui reconnaît le même langage que le langage généré par la grammaire régulière suivante. Le symbole initial est S. Décrivez le langage dans vos propres mots.

   $S \to aB$ $\quad$ $A \to aS$
   $S \to bA$ $\quad$ $A \to bS$
   $S \to \lambda$

3. Dessinez le diagramme de transitions d'un automate fini qui accepte le langage généré par la grammaire régulière ci-dessous (le symbole de départ est S). Décrivez le langage dans vos propres mots.

   $S \to \lambda$ $\quad$ $B \to bB$ $\quad$ $A \to aA$
   $S \to aA$ $\quad$ $B \to \lambda$ $\quad$ $A \to \lambda$
   $S \to bB$

4. Donnez une grammaire régulière qui génère le même langage que le langage accepté par l'automate fini suivant.



5. Trouvez une grammaire régulière qui génère l'ensemble des identificateurs Pascal.

6. Soit $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., E, +, -\}$. Donnez une grammaire régulière qui génère l'ensemble des nombres entiers et des nombres rationnels (avec ou sans exposant). Par exemple, 13, -8.45 et 44.888E-22 doivent être générés.

1.6 Expressions régulières

OBJECTIFS SPÉCIFIQUES

1. Décrire le langage représenté par une expression régulière donnée.
2. Construire une expression régulière représentant un langage donné.
3. Les grammaires régulières, les expressions régulières et les automates finis décrivent les mêmes langages : Passer d'une représentation à l'autre.

Nous allons maintenant développer une autre caractérisation des langages réguliers. Nous considérons d'abord les langages les plus simples qui peuvent être formés avec l'alphabet Σ : ce sont le langage vide $\emptyset$ et les langages ne contenant qu'une séquence d'un seul symbole, comme $\{a\}$ et $\{b\}$. Puis nous définirons des opérateurs qui nous permettront de combiner ces langages élémentaires pour former le langage $\{\lambda\}$ et d'autres langages plus complexes. Ces opérateurs sont l'union, la concaténation et la fermeture.

1.6.1 Union de deux langages

Soient deux langages L_1 et L_2. Étant donné qu'ils sont avant tout des ensembles, nous pouvons leur appliquer l'opérateur ensembliste qui fait l'union de ces deux ensembles. Cette union est bien sûr notée $L_1 \cup L_2$.

1.98 Exercice. Effectuez l'opération demandée.

1. $\{\lambda\} \cup \{a, ab, ba\} =$
2. $\{a^m b^n : m, n \in \mathbf{N} \wedge m < n\} \cup \{a^m b^n : m, n \in \mathbf{N} \wedge m > n\} =$

□

1.99 Théorème. *L'union de deux langages réguliers L_1 et L_2 est aussi un langage régulier.*

DÉMONSTRATION. En effet, soient

$$M_1 = (S_1, \Sigma, \rho_1, \iota_1, F_1) \text{ et } M_2 = (S_2, \Sigma, \rho_2, \iota_2, F_2),$$

où S_1 et S_2 sont disjoints, deux automates finis tels que $L_1 = L(M_1)$ et $L_2 = L(M_2)$. Construisons un automate fini

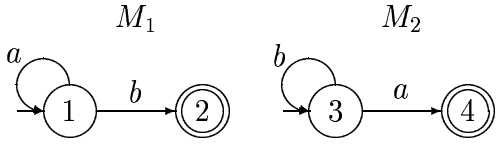
$$M = (S, \Sigma, \rho, \iota, F)$$

tel que $L(M) = L_1 \cup L_2$. Comme $L(M)$ est régulier (par définition), ceci démontrera que $L_1 \cup L_2$ est régulier. Définissons M comme suit :

- $S = S_1 \cup S_2 \cup \{\iota\}$ où ι est un nouvel état, i.e. $\iota \notin S_1 \cup S_2$.
- $\rho = \rho_1 \cup \rho_2 \cup \{(\iota, x, q) : (\iota_1, x, q) \in \rho_1 \vee (\iota_2, x, q) \in \rho_2\}$.
  C'est-à-dire que les transitions de M sont celles de M_1 et de M_2 plus des transitions partant de l'état initial de M simulant les transitions partant de l'état initial de M_1 et de l'état initial de M_2.
- $F = \begin{cases} F_1 \cup F_2 \cup \{\iota\} & \text{si } \iota_1 \in F_1 \text{ ou } \iota_2 \in F_2, \\ F_1 \cup F_2 & \text{sinon.} \end{cases}$
  En d'autres termes, les états finaux de M sont les états finaux de M_1 et de M_2, auxquels il faut ajouter l'état initial, si l'état initial de M_1 est final ou si l'état initial de M_2 est final.

De toute évidence, M accepte une chaîne si et seulement si elle est acceptée par M_1 ou par M_2. □

1.100 Exemple. Voici un exemple qui illustre la construction utilisée dans le théorème précédent. Soient les automates M_1 et M_2 suivants.



Construisons maintenant un automate M qui reconnaît $L(M_1) \cup L(M_2)$. L'ensemble des états de M est

$$S = \{\iota, 1, 2, 3, 4\}.$$

Les relations décrivant les transitions des deux automates ci-dessus sont

$$\rho_1 = \{(1, a, 1), (1, b, 2)\},$$
$$\rho_2 = \{(3, b, 3), (3, a, 4)\}.$$

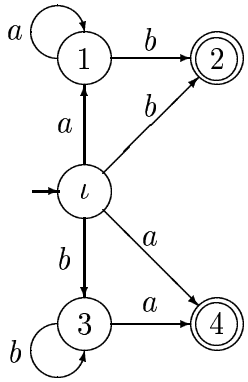
La relation de transition de M est donc

$$\begin{aligned} & \rho \\ = \ & \rho_1 \cup \rho_2 \cup \{(\iota, x, q) : (1, x, q) \in \rho_1 \vee (3, x, q) \in \rho_2\} \\ = \ & \{(1, a, 1), (1, b, 2), (3, b, 3), (3, a, 4), (\iota, a, 1), (\iota, b, 2), (\iota, b, 3), (\iota, a, 4)\}. \end{aligned}$$

Comme ni l'état initial de M_1 ni l'état initial de M_2 n'est final, l'ensemble des état finaux de M est

$$F = \{2, 4\}.$$

Le diagramme de transitions de M est donc

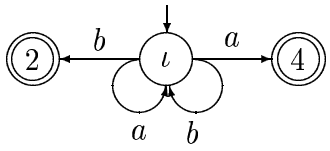


On a

$$\begin{aligned} L(M_1) &= \{a^n b : n \in \mathbf{N}\}, \\ L(M_2) &= \{b^n a : n \in \mathbf{N}\}, \\ L(M) &= L(M_1) \cup L(M_2) \\ &= \{s^n t : n \in \mathbf{N} \wedge ((s = a \wedge t = b) \vee (s = b \wedge t = a))\}. \end{aligned}$$

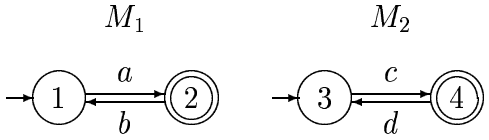
Les séquences de $L(M)$ ont soit la forme $a^n b$, soit la forme $b^n a$, pour $n \in \mathbf{N}$. La première transition de la machine M, effectuée à la lecture du premier symbole, décide de manière non déterministe si la suite de la reconnaissance s'effectue avec la sous-machine qui correspond à M_1 ou avec celle qui correspond à M_2.

Remarquons qu'il n'est pas correct de simplement fusionner les états initiaux. L'automate suivant, qui est le résultat d'une telle fusion, accepte la séquence aba, qui n'appartient pas à $L(M_1) \cup L(M_2)$.



□

1.101 Exercice. Donnez le diagramme de transitions d'un automate qui accepte l'union des langages acceptés par les automates suivants. L'alphabet est $\{a, b, c, d\}$.



□

1.6.2 Concaténation de deux langages

1.102 Définition. Soient L_1 et L_2 deux langages sur l'alphabet Σ. La *concaténation* de L_1 et L_2, dénotée par $L_1 \circ L_2$, est définie comme suit :

$$L_1 \circ L_2 = \{w_1 w_2 : w_1 \in L_1 \wedge w_2 \in L_2\}.$$

Autrement dit, $L_1 \circ L_2$ contient toutes les séquences formées par la concaténation d'une séquence de L_1 avec une séquence de L_2. □

1.103 Exemple. Si $L_1 = \{a, b\}$ et $L_2 = \{c, ad\}$ alors

$$L_1 \circ L_2 = \{ac, aad, bc, bad\}$$

□

1.104 Exercice. Effectuez les concaténations demandées.

1. $\{a, ab\} \circ \{b, ba\} =$
2. $\{b, ba\} \circ \{a, ab\} =$
3. $(\{a\} \circ \{aa, bb\}) \circ \{bb, ccc\} =$
4. $\{a\} \circ (\{aa, bb\} \circ \{bb, ccc\}) =$
5. $\{\lambda\} \circ \{a, ab, abc\} =$
6. $\{a, ab, abc\} \circ \{\lambda\} =$

□

1.105 Exercice. Vrai ou faux ? (L, L_1, L_2, L_3 sont des langages sur le même alphabet.)

1. Associativité de $\circ$: $L_1 \circ (L_2 \circ L_3) = (L_1 \circ L_2) \circ L_3$
2. Commutativité de $\circ$: $L_1 \circ L_2 = L_2 \circ L_1$
3. $\{\lambda\}$ élément neutre à gauche et à droite pour $\circ$: $\{\lambda\} \circ L = L \circ \{\lambda\} = L$

□

1.106 Théorème. *La concaténation de deux langages réguliers L_1 et L_2 est aussi un langage régulier.*

DÉMONSTRATION. Soient deux automates finis

$$M_1 = (S_1, \Sigma, \rho_1, \iota_1, F_1) \text{ et } M_2 = (S_2, \Sigma, \rho_2, \iota_2, F_2)$$

tels que $L_1 = L(M_1)$ et $L_2 = L(M_2)$. Construisons un automate fini

$$M = (S, \Sigma, \rho, \iota, F)$$

tel que $L(M) = L_1 \circ L_2$. Ceci démontrera que $L_1 \circ L_2$ est régulier. Définissons M comme suit :

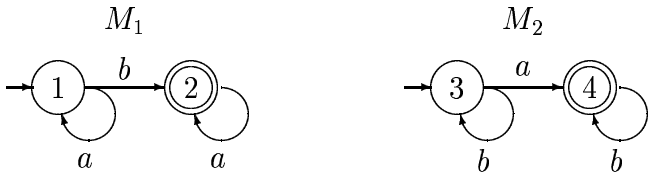
- $S = S_1 \cup S_2$.
- $\rho = \rho_1 \cup \rho_2 \cup \{(p, x, q) : p \in F_1 \wedge (\iota_2, x, q) \in \rho_2\}$.
  Les transitions de M sont celles de M_1 et M_2, plus celles qui sont nécessaires pour que M_1 passe le contrôle à M_2 ; ces dernières transitions simulent le comportement de l'état initial de M_2.
- L'état initial de M est l'état initial de M_1 : $\iota = \iota_1$.
- $F = \begin{cases} F_1 \cup F_2 & \text{si } \iota_2 \in F_2 \\ F_2 & \text{sinon.} \end{cases}$
  Les états finaux de M_2 sont des états finaux de M. Si l'état initial de M_2 est final, les états finaux de M_1 sont aussi des états finaux de M.

M accepte une chaîne en se comportant d'abord comme M_1, puis, après avoir atteint un état final de M_1, en se comportant comme M_2. □

1.107 Exemple. Soient les automates

$$M_1 = (\{1, 2\}, \{a, b\}, \rho_1, 1, \{2\}) \text{ et } M_2 = (\{3, 4\}, \{a, b\}, \rho_2, 3, \{4\}),$$

où ρ_1 et ρ_2 sont décrites par les diagrammes de transitions suivants :



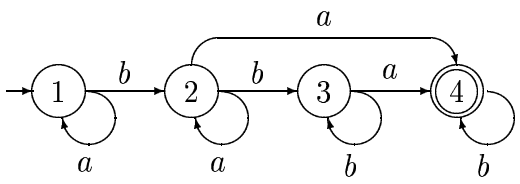
Construisons un automate M qui reconnaît $L(M_1) \circ L(M_2)$. La construction décrite dans la preuve du théorème 1.106 nous indique que

$$M = (\{1, 2, 3, 4\}, \{a, b\}, \rho, 1, \{4\}),$$

où

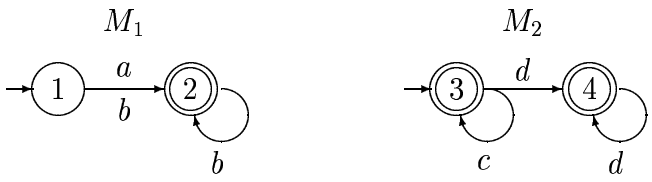
$$\begin{aligned} & \rho \\ = \ & \rho_1 \cup \rho_2 \cup \{(p, x, q) : p \in \{2\} \wedge (3, x, q) \in \rho_2\} \\ = \ & \{(1, a, 1), (1, b, 2), (2, a, 2)\} \cup \{(3, b, 3), (3, a, 4), (4, b, 4)\} \cup \{(2, b, 3), (2, a, 4)\} \\ = \ & \{(1, a, 1), (1, b, 2), (2, a, 2), (3, b, 3), (3, a, 4), (4, b, 4), (2, b, 3), (2, a, 4)\}. \end{aligned}$$

Le diagramme de transitions de M est donc



□

1.108 Exercice. Soient M_1 et M_2 les automates suivants.



Construisez un automate qui accepte $L(M_1) \circ L(M_2)$.

□

1.6.3 Fermeture d'un langage

Nous connaissons déjà l'opération de fermeture d'un alphabet Σ. Elle est dénotée par Σ^* et est l'ensemble de toutes les séquences finies de symboles de Σ. Nous allons étendre cette opération aux langages.

1.109 Définition. Soit L un langage. La *fermeture* de L, dénotée L^*, est l'ensemble des séquences formées en concaténant un nombre fini (0 ou plusieurs) de séquences de L. Autrement dit,

$$L^* = \{w_1 w_2 \cdots w_n : n \in \mathbf{N} \wedge w_i \in L \text{ pour tout } i = 1 \ldots n\}.$$

□

D'après cette définition, on remarque que $\lambda \in L^*$, pour tout langage L.

1.110 Exemple. Si $L_1 = \{a\}$ alors

$$L_1^* = \{\lambda, a, aa, aaa, aaaa, aaaaa, \ldots\}.$$

Si $L_2 = \{ab\}$ alors

$$L_2^* = \{\lambda, ab, abab, ababab, abababab, \ldots\}.$$

Si $L_3 = \{a, ab\}$ alors

$$L_3^* = \{\lambda, a, ab, aa, aab, aba, abab, aaa, aaba, aabab, abaa, \ldots\}.$$

□

1.111 Exercice. Énumérez les premières séquences des langages suivants.

1. $(\{ab\} \cup \{cd, \lambda\})^*$
2. $(\{ab\} \circ \{cd, \lambda\})^*$
3. $\emptyset^*$

□

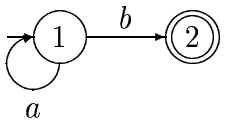
1.112 Théorème. *La fermeture d'un langage régulier L est un langage régulier.*

DÉMONSTRATION. Soit $M = (S, \Sigma, \rho, \iota, F)$ un automate fini tel que $L = L(M)$. Construisons un autre automate fini $M' = (S', \Sigma, \rho', \iota', F')$ tel que $L(M') = L^*$. M' est défini comme suit :

- $S' = S \cup \{\iota'\}$, où ι' est un nouvel état, i.e. $\iota' \notin S$.
- ι' est l'état initial.
- $F' = F \cup \{\iota'\}$. Ceci force l'acceptation de λ.
- $\rho' = \rho \cup \{(p, x, q) : p \in F' \wedge (\iota, x, q) \in \rho\}$.
  Ainsi, M' peut faire toutes les transitions de M, et en plus, à partir de ses états finaux, il simule le comportement de M dans l'état initial ι; ceci permet de revenir au « début » de l'automate pour reconnaître une nouvelle séquence de L.

□

1.113 Exemple. Voici un exemple qui illustre l'application du théorème précédent. Soit l'automate $M = (\{1, 2\}, \{a, b\}, \rho, 1, \{2\})$, où ρ est donnée par le diagramme de transitions qui suit.



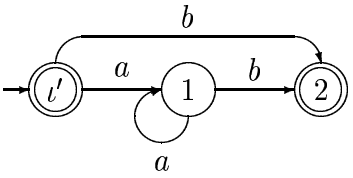
En utilisant la méthode du théorème 1.112, construisons un automate M' tel que $L(M') = (L(M))^*$. On a

$$M' = (\{1, 2, \iota'\}, \{a, b\}, \rho', \iota', \{\iota', 2\}),$$

où

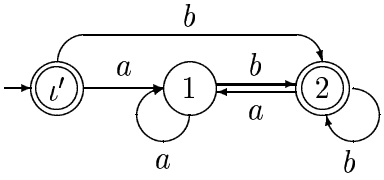
$$\begin{aligned} & \rho' \\ = \; & \rho \cup \{(p, x, q) : p \in \{\iota', 2\} \wedge (1, x, q) \in \rho\} \\ = \; & \{(1, a, 1), (1, b, 2)\} \cup \{(\iota', a, 1), (\iota', b, 2), (2, a, 1), (2, b, 2)\}. \end{aligned}$$

Pour mieux comprendre pourquoi M' reconnaît bien $(L(M))^*$, nous diviserons la construction du diagramme de transition de M' en deux étapes. À l'étape 1, ajoutons le nouvel état initial (toujours final) en plus des arcs qui en émergent.



On voit comment M', dans l'état initial ι', simule le comportement de M dans son état initial. Par exemple, à la lecture du symbole a, M' passe de l'état initial à l'état 1, tout comme M.

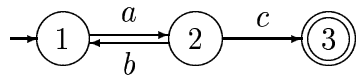
Ajoutons maintenant les arcs qui émergent de l'état final qui correspond à l'état final de M (c'est-à-dire l'état 2).



On voit comment M', dans l'état final 2, simule le comportement de M dans son état initial. Par exemple, dans l'état 2, à la lecture du symbole a, M' passe dans l'état 1, tout comme M. □

1.114 Exercice. Construisez l'automate qui accepte la fermeture du langage accepté par l'automate M suivant.

M :



□

1.6.4 Expressions régulières

Nous allons introduire le concept d'expression régulière, ce qui permettra de représenter les langages réguliers par des expressions régulières. Ces expressions sont essentiellement formées des symboles associés aux opérateurs précédemment définis sur l'ensemble des langages (union, concaténation, fermeture).

1.115 Définition. Une *expression régulière* sur un alphabet Σ se définit par les propositions récursives suivantes :

1. $\emptyset$ est une expression régulière.
2. x est une expression régulière, pour tout $x \in \Sigma$.
3. Si p et q sont des expressions régulières alors $(p \cup q)$ est une expression régulière.
4. Si p et q sont des expressions régulières alors $(p \circ q)$ est une expression régulière.
5. Si p est une expression régulière alors p^* est aussi une expression régulière.
6. Les seules expressions régulières sont celles qu'on peut construire selon 1 à 5.

□

Il faut respecter cette définition comme on respecte la syntaxe de Pascal lors de l'écriture d'un programme. Notez en particulier l'usage des parenthèses. Vous allez constater qu'elles s'accumulent rapidement lorsque les expressions s'allongent. On peut se donner des conventions qui permettent d'en supprimer une bonne partie, mais comme nous n'aurons pas à manipuler les expressions régulières intensivement, nous n'établirons pas de telles conventions.

1.116 Exercice. Soit $\Sigma = \{a, b, c\}$. Parmi les termes suivants, identifiez ceux qui sont des expressions régulières sur Σ. Lorsqu'un terme n'est pas une expression régulière, dites pourquoi.

1. $(a \cup (b \circ c))$
2. $[a \cup (b \circ c)]$
3. b^*
4. $(b)^*$
5. $(((a \cup b)^* \circ b)$
6. $(a \cup b)^* \circ b$

□

1.117 Définition. Soit une expression régulière r et un alphabet Σ. Le langage *représenté* par r, noté $L(r)$, est défini par les propositions récursives suivantes :

1. $L(\emptyset) = \emptyset$
2. Pour tout $x \in \Sigma$, $L(x) = \{x\}$
3. Si p et q sont deux expressions régulières alors $L((p \cup q)) = L(p) \cup L(q)$
4. Si p et q sont deux expressions régulières alors $L((p \circ q)) = L(p) \circ L(q)$
5. Si p est une expression régulière alors $L(p^*) = (L(p))^*$

□

1.118 Exemple. Appliquons la définition 1.117 afin de déterminer le langage représenté par des expressions régulières complexes (c'est-à-dire contenant plus d'un opérateur). L'alphabet est $\Sigma = \{a, b, c\}$.

1. $L((a \cup (b \circ c)))$
   $=$ ⟨ Clause 3 de la définition 1.117. ⟩
   $L(a) \cup L((b \circ c))$
   $=$ ⟨ Clause 4 de la définition 1.117. ⟩

$L(a) \cup (L(b) \circ L(c))$

$=$ ⟨ Clause 2 de la définition 1.117. ⟩

$\{a\} \cup (\{b\} \circ \{c\})$

$=$ ⟨ Définition de la concaténation de langages. ⟩

$\{a\} \cup \{bc\}$

$=$ ⟨ Définition de l'union de langages. ⟩

$\{a, bc\}$.

2. $L(((a \circ b)^* \cup c^*))$

$=$ ⟨ Clause 3 de la définition 1.117. ⟩

$L((a \circ b)^*) \cup L(c^*)$

$=$ ⟨ Clause 5 de la définition 1.117. ⟩

$(L((a \circ b)))^* \cup (L(c))^*$

$=$ ⟨ Clause 4 de la définition 1.117. ⟩

$(L(a) \circ L(b))^* \cup (L(c))^*$

$=$ ⟨ Clause 2 de la définition 1.117. ⟩

$(\{a\} \circ \{b\})^* \cup \{c\}^*$

$=$ ⟨ L'expression ci-dessus ne contient plus d'application de L à une expression régulière. Elle nous donne donc le langage représenté par l'expression de départ. Toutefois, nous pouvons simplifier encore un peu. Par définition de la concaténation, nous obtenons ce qui suit. ⟩

$\{ab\}^* \cup \{c\}^*$.

En utilisant les définitions de fermeture et d'union, nous aurions pu continuer jusqu'à obtenir

$$\{w^n : n \in \mathbf{N} \wedge (w = ab \vee w = c)\}.$$

Les deux réponses sont bonnes. Cependant, $\{ab\}^* \cup \{c\}^*$ se lit un peu plus facilement.

□

Attention : Il ne faut pas confondre expression régulière et langage. Une expression régulière sur un alphabet Σ est une séquence de symboles venant de $\Sigma \cup \{\cup, \circ, {}^*, \emptyset\}$ alors qu'un langage sur Σ est un ensemble de séquences de symboles pris dans Σ. Afin d'aider à faire cette distinction, nous avons choisi $\emptyset$ comme le symbole de l'expression régulière représentant le langage $\emptyset$ (remarquez la différence entre les deux symboles).

1.119 Exercice. En utilisant la définition 1.117, dites quel est le langage représenté par les expressions régulières suivantes. Donnez les étapes menant à cette description du langage (comme il est fait à l'exemple 1.118). Simplifiez la description du langage autant que possible.

1. $(a \circ (a \cup b))$
2. $((a \circ a)^* \cup (b \cup b)^*)$

□

1.120 Exercice. Pour chacun des langages suivants, donnez une expression régulière qui le représente.

1. $\{a^j b a^k : j, k \in \mathbf{N}^+\}$
2. $\{w^n : n \in \mathbf{N} \wedge (w = ab \vee w = ba)\}$

□

1.121 Remarque. L'expression qui représente le langage élémentaire $\{\lambda\}$ est $\emptyset^*$. En effet, $L(\emptyset^*) = (L(\emptyset))^* = \emptyset^* = \{\lambda\}$. □

Voici maintenant le théorème qui justifie le qualificatif *régulière* utilisé pour nommer le type d'expressions que nous venons de définir.

1.122 Théorème. *Soit un alphabet Σ. Les langages réguliers sur Σ sont précisément les langages représentables par les expressions régulières sur Σ.*

DÉMONSTRATION. Nous allons d'abord montrer que tout langage représentable par une expression régulière est régulier, puis que tout langage régulier est représentable par une expression régulière.

Partie 1

Montrons que si un langage est représentable par une expression régulière, alors c'est un langage régulier.

On sait que $\emptyset$ est un langage régulier, ainsi que $\{x\}$, pour tout $x \in \Sigma$. Par conséquent, $\emptyset$ et x représentent des langages réguliers. Les expressions régulières plus complexes sont construites à partir d'expressions plus simples en les combinant au moyen de symboles d'union, de concaténation ou de fermeture :

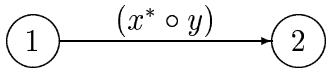
- Par le théorème 1.99, l'union de deux langages réguliers est un langage régulier. Par conséquent, $L((p \cup r))$, qui est égal à $L(p) \cup L(r)$ par la définition 1.117, est un langage régulier si $L(p)$ et $L(r)$ sont réguliers.
- Par le théorème 1.106, la concaténation de deux langages réguliers est un langage régulier. Par conséquent, $L((p \circ r))$, qui est égal à $L(p) \circ L(r)$ par la définition 1.117, est un langage régulier si $L(p)$ et $L(r)$ sont réguliers.
- Par le théorème 1.112, la fermeture d'un langage régulier est un langage régulier. Par conséquent, $L(p^*)$, qui est égal à $(L(p))^*$ par la définition 1.117, est un langage régulier si $L(p)$ est régulier.

On voit donc qu'il n'est pas possible de construire une expression régulière qui ne représente pas un langage régulier, puisque les expressions régulières élémentaires représentent des langages réguliers et que la combinaison d'expressions régulières représentant des langages réguliers est une expression régulière représentant un langage régulier.

Partie 2

Montrons que si un langage est régulier, alors il est représentable par une expression régulière.

Pour ce faire, nous associerons une expression régulière r à chaque automate fini M, de telle manière que $L(r) = L(M)$. Nous considérerons des diagrammes de transitions généralisés dont les arcs sont etiquetés par des expressions régulières. Voici un exemple d'une partie d'un tel diagramme.

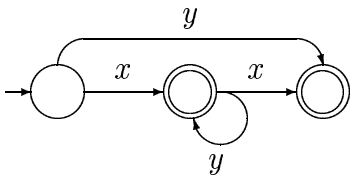


Pour passer de l'état 1 à l'état 2, l'automate doit lire une séquence de 0 ou plusieurs x, suivis d'un y.

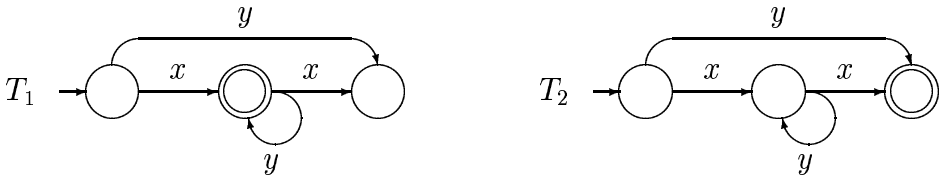
Si ces diagrammes généralisés peuvent être transformés en expressions régulières, les diagrammes usuels pourront l'être.

Soit T le diagramme de transitions de M. Pour trouver l'expression régulière r telle que $L(r) = L(M)$, il faut appliquer les règles suivantes jusqu'à ce que l'expression régulière soit trouvée.

1. Si T n'a pas d'état final, $r = \emptyset$.
2. Si T a n états finaux, $n > 1$, faire n copies $T_1, T_2, \ldots, T_n$ de T, chaque T_i ayant un seul état final différent des autres T_j. Évidemment, $L(T) = L(T_1) \cup \ldots \cup L(T_n)$. Par exemple, si T est le diagramme suivant,



on obtient



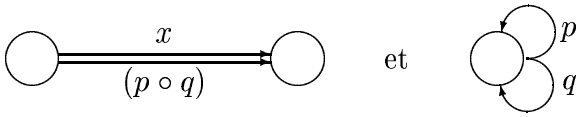
Après avoir fait cette transformation, il faut trouver des expressions régulières $r_1, r_2, \ldots, r_n$ telles que

$$L(r_i) = L(T_i), \text{ pour } i = 1 \ldots n.$$

On aura alors

$$r = ((r_1 \cup r_2) \cup \ldots \cup r_n).$$

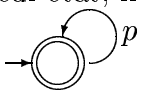
3. Si T a un seul état final.
   (a) Si, de l'état s_1 à l'état s_2, il y a k arcs étiquetés $r_1, r_2, \ldots, r_k$, les remplacer par un arc unique étiqueté $((r_1 \cup r_2) \cup \ldots \cup r_k)$. Par exemple, les deux parties de diagrammes qui suivent,



   deviennent, respectivement,

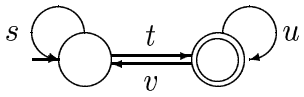
   (b) Si, entre chaque paire d'états, il y a au plus un arc dans chaque direction :

i. Si T a un seul état, il est nécessairement initial et final :

- Si T est  alors $r = p^*$.

- Si T est  alors $r = \emptyset^*$.

ii. Si T a deux états, l'un initial et l'autre final, il y a au plus quatre arcs :



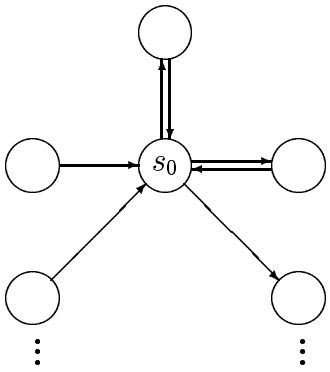
A. Si l'arc t est absent, $r = \emptyset$.

B. Si l'arc t est présent, il y a 8 cas. Dans chaque cas, l'expression régulière associée au diagramme est facile à déduire. Par exemple, si les seuls arcs présents sont s et t, il peut y avoir 0 ou plusieurs transitions par l'arc s, suivies d'une transition par l'arc t. L'expression est donc $(s^* \circ t)$.

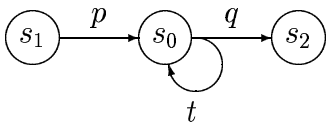
La table suivante donne une expression régulière pour chacun des 8 cas. En changeant la disposition des parenthèses, il est possible d'obtenir d'autres solutions correctes. Dans cette table, la lettre « p » signifie « présent » et la lettre « a » signifie « absent ».

s	u	v	r
p	p	p	$(((s^* \circ t) \circ u^*) \circ (v \circ ((s^* \circ t) \circ u^*))^*)$
p	p	a	$((s^* \circ t) \circ u^*)$
p	a	p	$((s^* \circ t) \circ (v \circ (s^* \circ t))^*)$
p	a	a	$(s^* \circ t)$
a	p	p	$((t \circ u^*) \circ (v \circ (t \circ u^*))^*)$
a	p	a	$(t \circ u^*)$
a	a	p	$(t \circ (v \circ t)^*)$
a	a	a	t

iii. Si T a au moins un état qui n'est ni initial, ni final : Soit s_0 l'un de ces états. La situation est la suivante (les boucles sont omises, parce que non pertinentes).



Enlever s_0 et ses arcs incidents. Pour chaque paire p, q d'arcs enlevés tels que

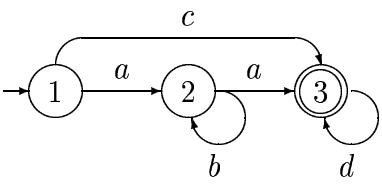


ajouter un arc de s_1 à s_2, avec l'expression
- $(p \circ q)$ si t est absent ;
- $((p \circ t^*) \circ q)$ si t est présent.

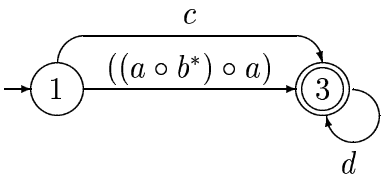
Remarque : Ce dernier cas permet d'enlever un état à l'automate. Un examen des cas traités montre que le nombre d'états doit diminuer jusqu'à un ou deux. Le processus termine donc.

□

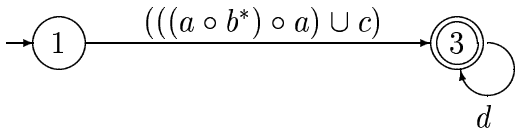
1.123 Exemple. Considérons le diagramme suivant.



Trouvons l'expression régulière associée à ce diagramme, selon la méthode de la partie 2 du théorème 1.122. L'état 2 n'est ni initial, ni final. Il faut donc appliquer le méthode du cas 3(b)iii, supprimer cet état 2 et ajouter un arc étiqueté $((a \circ b^*) \circ a)$ entre les états 1 et 3.



Il y a plus d'un arc entre les états 1 et 3. C'est le cas 3a du théorème. Faisons l'union des expressions de ces deux arcs et remplaçons les par un seul.

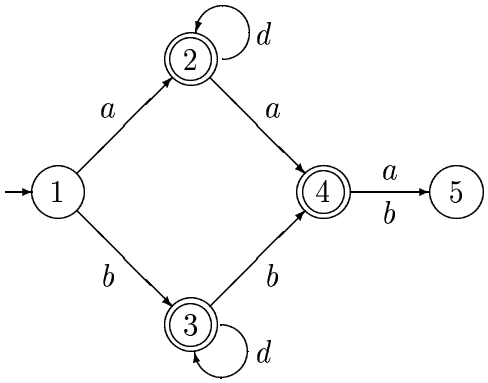


Il reste deux états ; l'un est initial (1) et l'autre final (2). L'arc (appelé t dans le théorème) entre l'état initial et l'état final est présent. On est donc dans le cas 3(b)iiB. Les arcs nommés s et v dans le théorème sont absents, mais l'arc u est présent. D'après la table, l'expression régulière correspondant au diagramme est

$$((((a \circ b^*) \circ a) \cup c) \circ d^*)$$

(il est aussi facile de la trouver directement par consultation du diagramme). □

1.124 Exercice. Donnez une expression régulière qui représente le langage accepté par l'automate M suivant.



□

1.6.5 Problèmes

1. Montrez que tout langage fini est régulier.
2. Décrivez (par des mots ou par une expression ensembliste) le langage représenté par chacune des expressions régulières suivantes.
   (a) $((c \cup b)^* \circ a)$
   (b) $((a \circ a^*) \circ (b \circ b^*))$
   (c) $((a \circ a^*) \cup (b \circ b^*))$
   (d) $((a^* \circ b^*) \circ c^*)$
3. Trouvez un diagramme de transitions qui accepte le langage généré par la grammaire régulière suivante. (Le symbole de départ est S.) Ensuite, trouvez une expression régulière représentant le même langage.

   $S \to aN$ $\quad$ $M \to cN$ $\quad$ $N \to bM$
   $S \to a$ $\quad$ $M \to c$ $\quad$ $N \to b$

4. Trouvez une expression régulière qui représente l'intersection des langages représentés par chacune des paires d'expressions régulières qui suivent.
   (a) $(a \cup b^*)$ et $(a \cup b)^*$
   (b) $(a \circ (a \cup b)^*)$ et $((a \cup b)^* \circ b)$
   (c) $(((a \cup b) \circ b) \circ (a \cup b)^*)$ et $(b \circ (a \cup b)^*)$
   (d) $((a \circ b)^* \circ (a \cup \emptyset))$ et $((a \cup b) \circ (a \circ b)^*)$
5. Pour chacune des grammaires régulières suivantes, donnez une expression régulière qui représente le langage généré par la grammaire. Toutes ces grammaires ont S pour symbole de départ (axiome). Cela peut se faire directement, en analysant le langage généré par la grammaire. Cela peut aussi se faire en trouvant l'automate qui accepte le langage généré par la grammaire, puis en trouvant l'expression régulière associée à l'automate.

   (a) $S \to aT$ $\quad$ $S \to bS$ $\quad$ $T \to aT$
   $S \to a$ $\quad$ $S \to b$ $\quad$ $T \to a$

   (b) $S \to bS$ $\quad$ $T \to bT$
   $S \to aT$ $\quad$ $T \to aS$
   $S \to b$ $\quad$ $T \to a$

   (c) $S \to aB$ $\quad$ $A \to aS$
   $S \to bA$ $\quad$ $A \to bS$
   $S \to \lambda$

   (d) $S \to aB$ $\quad$ $A \to aB$ $\quad$ $B \to bS$
   $S \to bA$ $\quad$ $A \to a$

6. Trouvez l'automate reconnaissant le langage représenté par l'expression $((a \cup b^*) \circ c)$. Il est possible de le faire directement en analysant le langage représenté par l'expression. Il est aussi possible de construire trois automates élémentaires qui acceptent a, b et c, puis de les combiner en en faisant l'union, la concaténation ou la fermeture (voir les théorèmes 1.99, 1.106 et 1.112).
7. Montrez que si L_1 et L_2 sont deux langages réguliers, alors $L_1 \cap L_2$ est aussi un langage régulier.
8. Montrez que si L_1 et L_2 sont deux langages réguliers, alors le langage $L_1 - L_2$ est aussi régulier.

Utilisons les termes correctement

- Un automate reconnaît ou accepte une séquence.
- Un automate reconnaît ou accepte un langage.
- Une grammaire génère une séquence.
- Une grammaire génère un langage.
- Une expression régulière représente une séquence.
- Une expression régulière représente un langage.
- Un langage contient une séquence.
- Une séquence appartient à un langage.

Chapitre 2

Automates à pile, langages non contextuels

OBJECTIF GÉNÉRAL

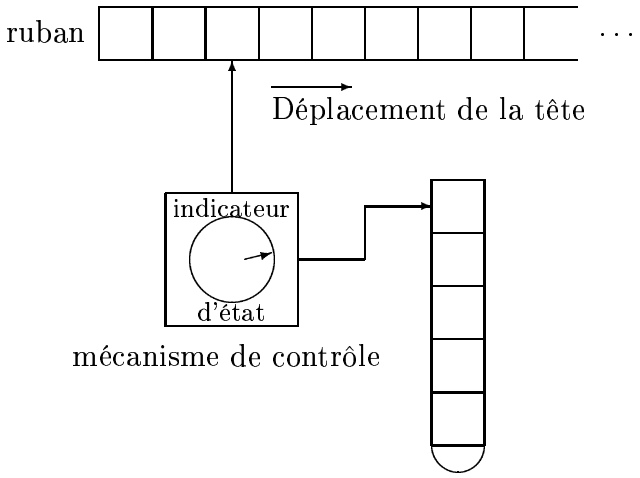
1. Comprendre les capacités et les limites des automates à pile en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires non contextuelles.

2.1 Automates à pile

OBJECTIFS SPÉCIFIQUES

1. Démontrer le fonctionnement d'un automate à pile donné en énumérant les configurations du ruban et celles de la pile ainsi que les états consécutifs à une configuration initiale donnée.
2. Démontrer l'acceptation ou le rejet d'une séquence de symboles donnée par un automate à pile donné.
3. Transformer un automate qui accepte sans vider sa pile en automate qui vide sa pile avant d'accepter.
4. Décrire le langage accepté par un automate à pile donné.
5. Construire un automate à pile acceptant un langage donné.

Nous allons étudier une classe de machines plus puissantes : les automates à pile. Elles sont capables de reconnaître les expressions parenthésées et les imbrications de blocs `begin-end`. Pour augmenter la puissance des automates finis, on leur donne de la mémoire en leur ajoutant une pile.



La pile de l'automate a son propre alphabet qui peut être différent de l'alphabet d'entrée. À part la pile, l'automate à pile est comme l'automate fini.

Les opérations que peut effectuer un automate à pile sont les suivantes :

- lire un symbole (et avancer la tête de lecture)
- dépiler un symbole
- empiler un symbole

- changer d'état

Pour représenter une transition d'un automate à pile, on utilise la notation

$$(p, x, y; q, z),$$

où p, x, y, q et z sont respectivement l'état courant, le symbole à l'entrée, le symbole dépilé, le nouvel état et le symbole empilé. Notez que le y est dépilé avant que le z soit empilé. L'automate n'est pas obligé de faire à la fois une lecture, un empilement et un dépilement à chaque transition. Si aucun symbole n'est lu, on met un λ à la place du x. Si aucun symbole n'est dépilé, on met un λ à la place du y. Si aucun symbole n'est empilé, on met un λ à la place du z.

2.1 Exemple. La transition $(1, a, t; 2, b)$ signifie que l'automate passe de l'état 1 à l'état 2, en lisant un a sur le ruban, en dépilant un t et en empilant un b.

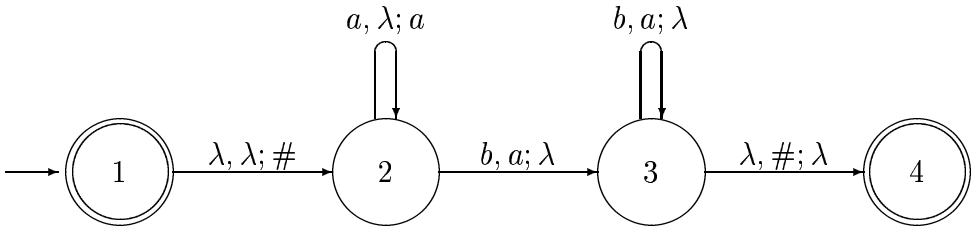
$(p, \lambda, \lambda; q, \lambda)$ signifie que l'automate passe de l'état p à l'état q (aucune lecture, ni empilement, ni dépilement).

$(p, \lambda, 2; q, \lambda)$ signifie que l'automate passe de l'état p à l'état q en enlevant le symbole 2 de la pile.

$(p, a, \lambda; q, c)$ indique que l'automate passe de l'état p à l'état q en lisant du ruban le symbole a et en plaçant c sur la pile. □

Les diagrammes de transitions des automates à pile sont comme ceux des automates finis sauf que l'étiquette d'un arc est plus complexe. Ces étiquettes ont la forme $x, y; z$, où x est le symbole lu sur le ruban, y est le symbole dépilé et z est le symbole empilé.

2.2 Exemple. Voici le diagramme de transitions d'un automate à pile :



En passant de l'état 1 à l'état 2, l'automate empile un #. En passant de l'état 2 à l'état 3, il lit un b et dépile un a. □

2.3 Exercice. Que fait l'automate de l'exemple 2.2 en passant

1. de l'état 2 à l'état 2 ?
2. de l'état 3 à l'état 3 ?

3. de l'état 3 à l'état 4 ?

□

Allons y maintenant pour la définition formelle d'un automate à pile.

2.4 Définition. Un *automate à pile* est un sixtuplet $(S, \Sigma, \Gamma, T, \iota, F)$ où
- S est un ensemble fini d'états,
- Σ est l'alphabet d'entrée (alphabet du ruban),
- Γ est l'alphabet de la pile,
- T est un ensemble fini de transitions,
- $\iota \in S$ est l'état initial,
- $F \subseteq S$ est l'ensemble des états finaux.

L'automate à pile est démarré dans l'état initial avec la pile vide. Il accepte la séquence d'entrée ssi il peut atteindre un état final après l'avoir lue en entier. Le langage accepté par un automate à pile M est noté $L(M)$; c'est l'ensemble des séquences acceptées (c'est-à-dire que $L(M) = \{w : w \text{ est acceptée }\}$). □

Dans cette définition, les mots *peut* et *après* sont utilisés pour les raisons suivantes.
- peut : Puisque la définition d'un automate à pile n'exige pas que l'ensemble T des transitions soit une fonction, les automates à pile peuvent être non déterministes.
- après : Il n'est pas nécessaire que l'automate à pile atteigne un état final tout de suite après avoir lu le dernier symbole d'entrée. Après avoir terminé la lecture de la séquence d'entrée, l'automate peut effectuer des transitions en manipulant sa pile et atteindre de cette façon un état accepteur.

2.5 Exemple. Soit l'automate à pile $M = (\{1,2,3,4\}, \{a,b\}, \{a,b,\#\}, T, 1, \{1,4\})$, où T est l'ensemble de transitions

$$\{(1,\lambda,\lambda;2,\#), (2,a,\lambda;2,a), (2,b,a;3,\lambda), (3,b,a;3,\lambda), (3,\lambda,\#;4,\lambda)\}.$$

Le diagramme de transitions correspondant est celui de l'exemple 2.2. On pourrait aussi considérer que son alphabet de pile est $\{a,\#\}$, puisqu'il n'utilise pas de b sur la pile. Cet automate accepte la séquence λ, car l'état initial est final. Il accepte aussi la séquence $aabb$ en passant par la suite de configurations que voici.

état	ruban	pile
1	$\underline{a}abb$	
2	$\underline{a}abb$	$\#$
2	$a\underline{a}bb$	$\#a$
2	$aa\underline{b}b$	$\#aa$
3	$aab\underline{b}$	$\#a$
3	$aabb\_$	$\#$
4	$aabb\_$	

Voyons ce qui se passe si la séquence d'entrée est aab.

état	ruban	pile
1	$\underline{a}ab$	
2	$\underline{a}ab$	$\#$
2	$a\underline{a}b$	$\#a$
2	$aa\underline{b}$	$\#aa$
3	$aab\underline{\ }$	$\#a$

Aucune transition n'est possible, car il n'y a plus de b à lire et il n'est pas possible de dépiler le $\#$, car il y a un a par dessus. La séquence aab est rejetée, car l'état 3 n'est pas final.

En dernier lieu, voici le traitement réservé à $aabbb$.

état	ruban	pile	
1	$\underline{a}abbb$		
2	$\underline{a}abbb$	$\#$	
2	$a\underline{a}bbb$	$\#a$	
2	$aa\underline{b}bb$	$\#aa$	
3	$aab\underline{b}b$	$\#a$	
3	$aabb\underline{b}$	$\#$	
4	$aabb\underline{b}$		seule transition possible

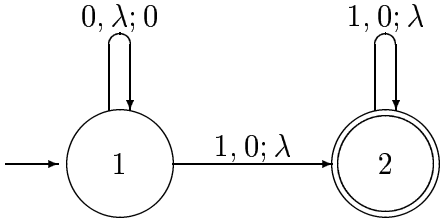
La séquence d'entrée n'est pas toute lue et ne peut l'être (il n'y a pas d'autre choix de chemin). Elle est donc rejetée.

Le langage accepté par M est

$$\{a^n b^n : n \in \mathbf{N}\}.$$

Comme nous l'avons vu au chapitre 1, aucun automate fini ne peut accepter ce langage. □

2.6 Exercice. Soit le diagramme de transitions d'un automate à pile :



1. Quelle est la définition formelle de cet automate (c'est-à-dire quels sont ses paramètres $S, \Sigma, \Gamma, T, \iota, F$) ?

2. Donnez la suite des configurations lorsque la séquence d'entrée est
   (a) 0011

   (b) 011

   (c) 001

3. Quel langage accepte-t-il ?

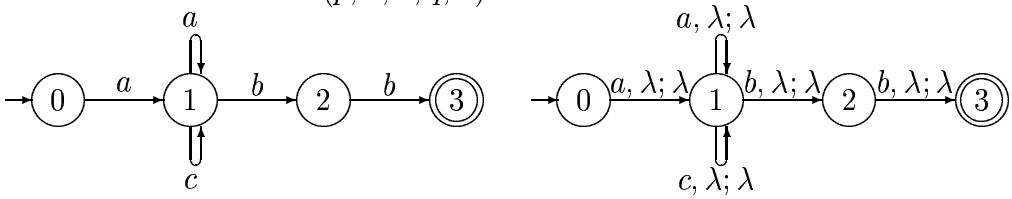
□

Si toutes les transitions d'un automate à pile ont la forme $(p, x, \lambda; q, \lambda)$, cet automate à pile se comporte comme un automate fini (puisqu'il ne manipule pas la pile). Donc

$$\{\text{langages réguliers}\} \subseteq \{L(M) : M \text{ est un automate à pile}\}.$$

Comme nous l'avons vu à l'exemple 2.5, il existe un automate à pile qui accepte $\{a^n b^n : n \in \mathbf{N}\}$, alors qu'aucun automate fini ne peut le faire. Les automates à pile sont donc strictement plus puissants que les automates finis.

2.7 Exemple. Voici un automate fini et un automate à pile équivalent dont toutes les transitions ont la forme $(p, x, \lambda; q, \lambda)$:



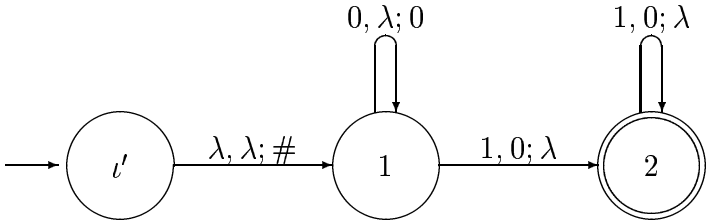
□

Considérons l'automate de l'exercice 2.6. Si on lui soumet à l'entrée la chaîne 0001, l'automate accepte cette dernière en laissant deux 0 sur la pile. Pour certaines applications où l'utilisation des automates à pile est nécessaire, ce type de comportement pourrait être jugé inacceptable. Qu'advient-il de la classe des langages reconnus par les automates à pile si on les oblige à vider leur pile avant d'accepter ? Le théorème suivant répond à cette question.

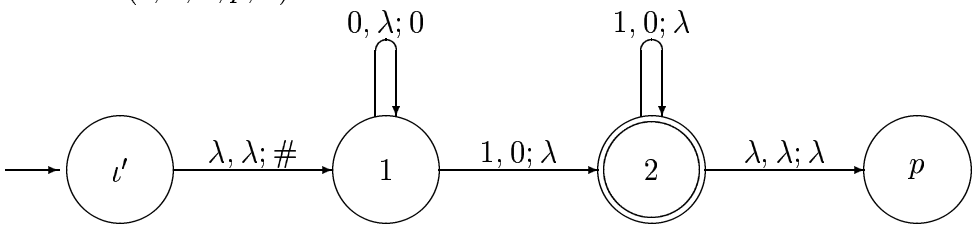
2.8 Théorème. *Soit $(S, \Sigma, \Gamma, T, \iota, F)$ un automate à pile qui peut accepter des séquences sans vider sa pile. Il existe un automate à pile $M' = (S', \Sigma, \Gamma', T', \iota', F')$ qui vide toujours sa pile avant d'accepter et tel que $L(M) = L(M')$.*

DÉMONSTRATION. (Cette démonstration est combinée à un exemple indiquant comment effectuer la transformation de l'automate de l'exercice 2.6. Nous prenons $\Gamma = \{0, 1\}$.) On transforme M en M' de la façon suivante :

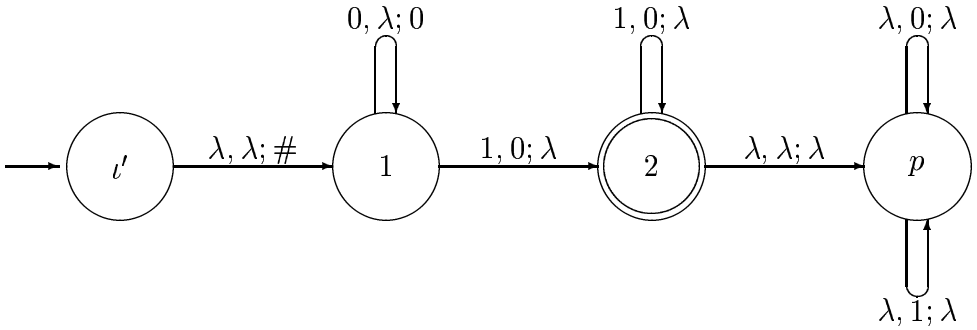
1. Ajouter un nouvel état initial ι' et une transition $(\iota', \lambda, \lambda; \iota, \#)$ où $\# \notin \Gamma$.



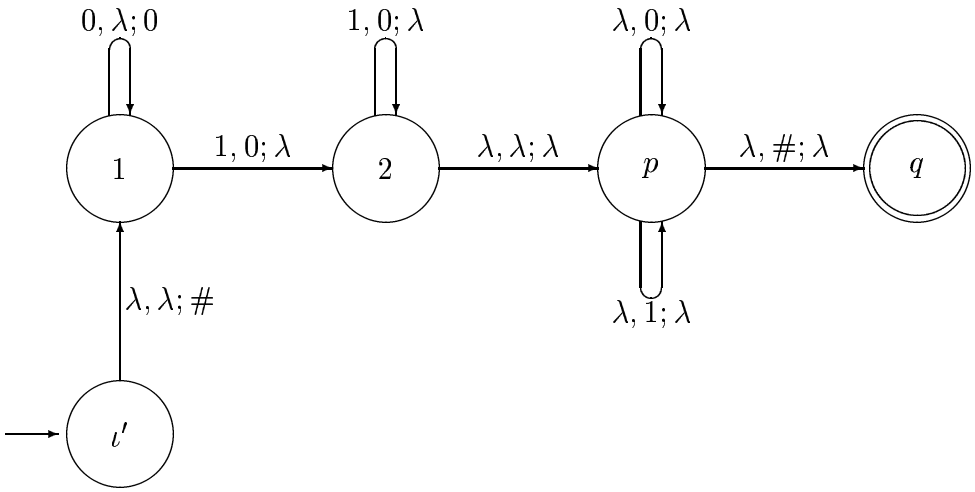
2. Ensuite, ajouter un nouvel état p. Puis, pour tous les $s \in F$, ajouter une transition $(s, \lambda, \lambda; p, \lambda)$.



3. Pour tout $z \in \Gamma$, ajouter les transitions $(p, \lambda, z; p, \lambda)$ (rappel : $\# \notin \Gamma$).



4. Ajouter un nouvel état q qui devient le seul état final, ainsi qu'une transition $(p, \lambda, \#; q, \lambda)$.

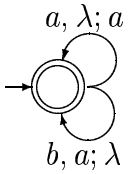


Le fonctionnement de l'automate M' se résume de la façon suivante. D'abord, M' marque la pile avec le symbole $\#$, puis fait comme M.

- Si M lit toute la chaîne et la rejette en terminant dans l'état 1, M' fait de même.
- Si M termine dans l'état 2 sans lire toute la chaîne, il la rejette. M' fait la transition de 2 vers p, vide la pile (sauf $\#$), puis fait la transition de p vers q (en dépilant $\#$) et arrête en rejetant (car il n'a pas lu toute la chaîne d'entrée).
- Si M finit dans l'état 2 avec toute la séquence lue, M' fait la même chose puis va en p, dépile jusqu'au symbole $\#$, puis va à l'état q en dépilant $\#$ et accepte avec sa pile vide.

□

2.9 Exercice. Voici le diagramme de transitions d'un automate à pile.



Transformez cet automate pour qu'il accepte le même langage en vidant toujours sa pile lorsqu'il accepte.

□

2.10 Remarque. Par la suite, à moins d'indication contraire, nous supposerons que les automates peuvent accepter sans vider leur pile. □

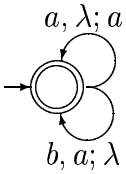
2.11 Exercice. Construisez un automate à pile acceptant le langage

$$\{0^n 1^m 0^n : m, n \in \mathbf{N}\}.$$

□

2.1.1 Problèmes

1. Quel est le langage accepté par l'automate à pile dont le diagramme de transitions est le suivant ?



2. Construisez un automate à pile qui reconnaît le langage

$$\{a^r b^s c^t : r \in \mathbf{N} \wedge s \in \mathbf{N} \wedge t \in \mathbf{N} \wedge s = r + t\}.$$

2.2 Grammaires non contextuelles

OBJECTIFS SPÉCIFIQUES

1. Décrire le langage généré par une grammaire non contextuelle donnée.
2. Construire une grammaire non contextuelle générant un langage donné.
3. Passer d'une grammaire non contextuelle à un automate à pile.
4. Convertir une grammaire non contextuelle quelconque en grammaire sous forme normale de Chomsky.
5. Donner l'arbre de dérivation d'une séquence donnée par une grammaire non contextuelle donnée.

Dans cette section, nous introduisons des grammaires plus puissantes que les grammaires régulières en levant les restrictions imposées au côté droit des productions de ces dernières (voir définition 1.88). Nous verrons que ces grammaires, appelées grammaires non contextuelles, peuvent générer exactement les langages reconnaissables par les automates à pile. En particulier, elles peuvent générer des expressions arithmétiques dont les parenthèses sont correctement équilibrées. Ce type de grammaire est le plus utilisé en informatique. En effet, les grammaires BNF[1], qui sont utilisées pour définir la syntaxe des langages de programmation, sont des grammaires équivalentes aux grammaires non contextuelles.

2.12 Définition. Une *grammaire non contextuelle* (ou *indépendante du contexte*) est une grammaire de phrases (voir définition 1.78) dont le côté gauche de chaque règle consiste en un seul symbole non terminal. □

Nous avons vu au chapitre précédent que le côté droit des règles des grammaires régulières est constitué soit d'un terminal, d'un terminal suivi d'un non terminal, ou de la chaîne vide. Dans le cas des grammaires non contextuelles, le côté droit des règles peut être n'importe quel arrangement de terminaux et de non terminaux. Par contre, les deux types de grammaires ont les mêmes restrictions sur le côté gauche. Les grammaires régulières sont donc un sous ensemble strict de l'ensemble des grammaires non contextuelles.

2.13 Exemple. Voici une grammaire non contextuelle $G = (V, T, S, R)$, où

1. $V = \{$ *expr, instr, liste_instr* $\}$
2. $T = \{$ **id**, **:=**, **si**, **alors**, **tant**, **que**, **faire**, **début**, **fin**, **;**, **a** $\}$
3. $S =$ *instr*

[1] Backus-Naur Form ou Backus Normal Form

4. R :
$$\begin{aligned}
\textit{instr} &\rightarrow \textbf{id} := \textit{expr}\\
\textit{instr} &\rightarrow \textbf{si } \textit{expr} \textbf{ alors } \textit{instr}\\
\textit{instr} &\rightarrow \textbf{tant que } \textit{expr} \textbf{ faire } \textit{instr}\\
\textit{instr} &\rightarrow \textbf{début } \textit{liste_instr} \textbf{ fin}\\
\textit{liste_instr} &\rightarrow \textit{instr} \textbf{ ; } \textit{liste_instr}\\
\textit{liste_instr} &\rightarrow \lambda\\
\textit{expr} &\rightarrow \textbf{a}
\end{aligned}$$

□

Le terme « non contextuelle » vient du fait qu'étant donné que le côté gauche d'une règle est constitué uniquement d'un non terminal, la règle peut être appliquée sans égard au contexte dans lequel se retrouve le non terminal. Par exemple, grâce à la règle $Z \rightarrow aAb$, Z peut toujours être remplacé peu importe le contexte où il se trouve. Par contre, considérons la règle $aZb \rightarrow aAb$. Étant donnée une telle règle, Z peut être remplacé par A seulement s'il est précédé par un a et suivi par un b.

Les grammaires non contextuelles génèrent des chaînes par dérivation tout comme les grammaires régulières. Mais dans le cas des grammaires non contextuelles, au cours du processus de dérivation, la possibilité peut s'offrir à nous de choisir parmi plusieurs non terminaux à remplacer.

2.14 Exemple. Pour illustrer le phénomène, dérivons une chaîne en appliquant les règles de la grammaire G ci-dessous.

$$\begin{array}{lll}
S \rightarrow zMNz & M \rightarrow aMa & N \rightarrow bNb\\
 & M \rightarrow z & N \rightarrow z
\end{array}$$

$$S \Rightarrow zMNz \Rightarrow zaMaNz \Rightarrow zaMazz \Rightarrow zazazz$$

On voit qu'à la deuxième étape, le choix s'offre de remplacer le non terminal M ou le non terminal N. Alors, est-ce que la chaîne obtenue aurait été différente si nous avions appliqué les mêmes règles mais dans un ordre différent ? La réponse est non et nous verrons plus loin pourquoi. □

2.15 Exercice. Décrivez par une expression ensembliste le langage généré par la grammaire de l'exemple 2.14.

□

2.16 Définition. Une *dérivation à gauche* s'effectue en remplaçant toujours, à chaque étape du processus de dérivation, le non terminal le plus à gauche. □

2.17 Exemple. Étant donnée la grammaire de l'exemple 2.14, voici une dérivation à gauche :

$$S \Rightarrow zMNz \Rightarrow zaMaNz \Rightarrow zazaNz \Rightarrow zazabNbz \Rightarrow zazabzbz$$

□

2.18 Définition. Une *dérivation à droite* s'effectue en remplaçant toujours, à chaque étape du processus de dérivation, le non terminal le plus à droite. □

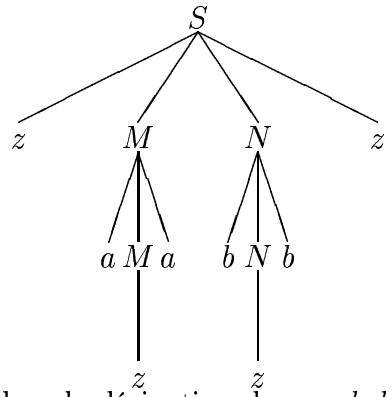
2.19 Exemple. Étant donnée la grammaire de l'exercice 2.14, voici une dérivation à droite :

$$S \Rightarrow zMNz \Rightarrow zMbNbz \Rightarrow zMbzbz \Rightarrow zaMabzbz \Rightarrow zazabzbz$$

Cette séquence est la même que celle qui a été obtenue à l'exemple 2.17. Elle est ici le résultat de l'application des mêmes règles, mais dans un ordre différent. □

D'autres dérivations sont possibles. Mais toutes les dérivations donnent les mêmes séquences terminales. Cela est assez facile à voir en considérant l'arbre de dérivation.

La racine de l'arbre de dérivation est l'axiome de la grammaire. Les non terminaux forment les noeuds internes de l'arbre et les terminaux forment les feuilles.



Arbre de dérivation de $zazabzbz$

En regardant l'arbre, on voit que le choix du symbole non terminal à réécrire n'affecte pas la séquence terminale dérivée puisque les branches de l'arbre sont indépendantes. Donc, l'ordre d'application des règles de production n'affecte pas le résultat final.

2.20 Exercice. Soit la grammaire $G = (\{S\}, \{+, -, *, /, (,), a\}, S, R)$ où R est l'ensemble des règles suivantes :

$$\begin{array}{lll} S \to S + S & S \to S * S & S \to (S) \\ S \to S - S & S \to S/S & S \to a \end{array}$$

Donnez une dérivation à gauche, une dérivation à droite et l'arbre de dérivation de la séquence $a * (a + a)$.

□

Même si l'ordre d'application des règles de production n'affecte pas le résultat final, il peut arriver que, pour une séquence terminale donnée, des choix différents de règles résultent en des arbres différents. Ce cas est suffisamment important pour que l'on donne un nom aux grammaires qui le permettent.

2.21 Définition. Une grammaire est dite *ambiguë* si elle permet plus d'un arbre de dérivation pour une séquence terminale donnée. □

2.22 Exercice. Montrez que la grammaire G ci-dessous est ambiguë en produisant la chaîne

si condition alors si condition alors instruction sinon instruction

par des dérivations dont les arbres sont différents.

$G = (\{S\}, \{$ **condition, si, alors, sinon, instruction** $\}, S, R)$,

où R est l'ensemble des règles suivantes.

(1) $S \rightarrow$ **si condition alors** S
(2) $S \rightarrow$ **si condition alors** S **sinon** S
(3) $S \rightarrow$ **instruction**

Quel est le résultat du programme suivant, selon l'arbre de dérivation utilisé pour analyser l'instruction conditionnelle ?

```
x :=1 ; si x>5 alors si x<10 alors x :=x+1 sinon x :=x-1
```

□

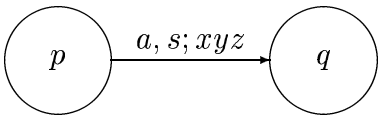
2.23 Définition. Un langage est dit *non contextuel* s'il est généré par une grammaire non contextuelle. □

2.24 Exercice. Construisez une grammaire non contextuelle qui génère le langage $\{a^m b^n c^m : m \in \mathbf{N} \wedge n \in \mathbf{N}^+\}$.

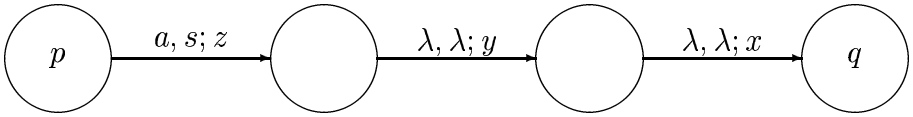
□

2.25 Notation. Afin de démontrer qu'il existe toujours un automate à pile qui accepte le langage généré par une grammaire non contextuelle donnée, nous généralisons la notation utilisée pour décrire les transitions d'un automate à pile. Les transitions peuvent maintenant avoir la forme $(p, a, s; q, xyz)$, c'est-à-dire que plusieurs symboles

peuvent être empilés à la fois. La section d'automate correspondant à une telle transition est la suivante.



Un tel diagramme est en fait une abréviation du diagramme qui suit.



De ce deuxième diagramme, on voit que l'empilement se fait dans l'ordre z, y, x c'est-à-dire $\begin{matrix} x \\ y \\ z \end{matrix}$. □

2.26 Théorème. *Pour toute grammaire non contextuelle G, il y a un automate à pile M tel que $L(G) = L(M)$.*

DÉMONSTRATION. On a $G = (V, T, S, R)$ non contextuelle. On cherche un automate à pile $M = (S_M, \Sigma, \Gamma, T_M, \iota, F)$ tel que $L(G) = L(M)$. Définissons

$$\begin{aligned} S_M &= \{\iota, p, q, f\} \text{ où } \iota \text{ est l'état initial} \\ F &= \{f\} \\ \Sigma &= T \\ \Gamma &= V \cup T \cup \{\#\} \text{ où } \# \notin V \text{ et } \# \notin T \\ T_M &= \{(\iota, \lambda, \lambda; p, \#), (p, \lambda, \lambda; q, S), (q, \lambda, \#; f, \lambda)\} \\ &\quad \cup \{(q, \lambda, N; q, w) : N \to w \text{ est une règle de } G\} \\ &\quad \cup \{(q, x, x; q, \lambda) : x \in T\} \end{aligned}$$

Un automate à pile construit de cette manière analyse la séquence d'entrée en marquant d'abord le bas de la pile avec le symbole $\#$, puis en y empilant l'axiome de la grammaire tout en passant dans l'état q. À partir de là, tant que le $\#$ ne refait pas surface, l'automate

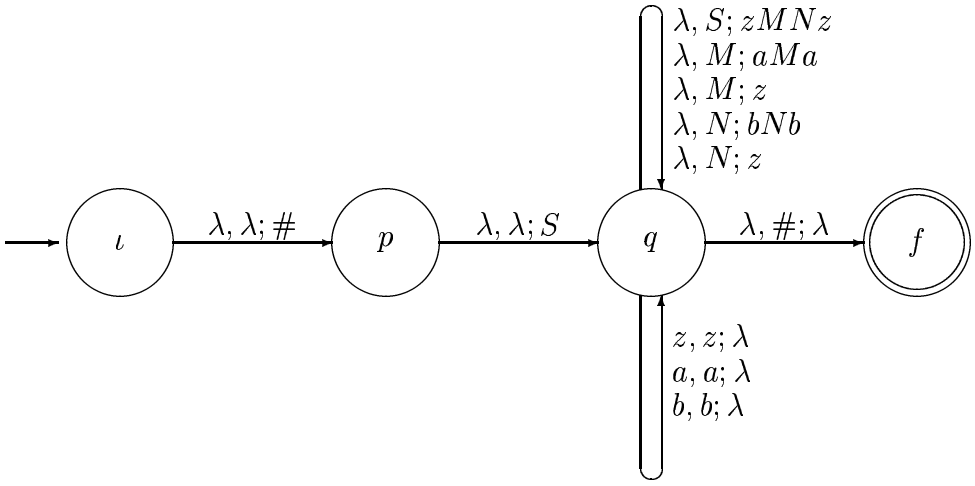
- ou bien dépile un symbole non terminal et le remplace par le côté droit d'une règle contenant ce non terminal à gauche (s'il y a plusieurs règles possibles, le choix est fait de manière non déterministe),
- ou bien dépile un symbole terminal tout en lisant le même symbole à l'entrée (si le symbole à lire est différent, l'automate n'a plus de transition possible).

Lorsque le $\#$ revient au sommet de la pile, M passe dans l'état f, qui est final ; la séquence d'entrée est alors acceptée si elle est toute lue.

Remarquons que lorsque le côté droit d'une règle est empilé, le symbole non terminal de gauche de cette règle (s'il y en a un) est plus près du sommet de la pile que les autres symboles non terminaux (s'il y en a d'autres). Ce sera donc le premier non terminal a être remplacé. Par conséquent, l'automate analyse la séquence d'entrée en faisant une dérivation à gauche basée sur les règles de la grammaire donnée. Mais s'il existe une dérivation de la séquence d'entrée, il existe une dérivation à gauche. Ainsi, l'automate M accepte exactement le même langage que le langage généré par la grammaire G. □

La meilleure manière de se convaincre de la validité de ce théorème est de considérer un exemple et de faire la construction proposée.

2.27 Exemple. Voici un automate à pile qui accepte le langage généré par la grammaire de l'exemple 2.14. Il est construit selon la méthode du théorème 2.26.



L'acceptation de la chaîne *zazabzbz* s'effectue de la façon suivante. Remarquez comment elle correspond à la dérivation à gauche

$$S \Rightarrow zMNz \Rightarrow zaMaNz \Rightarrow zazaNz \Rightarrow zazabNbz \Rightarrow zazabzbz$$

Dans le tableau de configurations qui suit, la pile croît vers la gauche, contrairement à la convention que nous avons utilisée à la section précédente (voir l'exemple 2.5). Nous utilisons cette nouvelle convention parce que l'empilement vers la gauche permet de mieux voir le côté droit des règles dans la pile (évidemment, on pourrait faire une

pile verticale, ce qui éliminerait ce problème ; mais cela prend plus de place).

Étape	Entrée	État	Pile
0	$\underline{z}azabzbz$	ι	λ
1	$\underline{z}azabzbz$	p	$\#$
2	$\underline{z}azabzbz$	q	$S\#$
3	$\underline{z}azabzbz$	q	$zMNz\#$
4	$z\underline{a}zabzbz$	q	$MNz\#$
5	$z\underline{a}zabzbz$	q	$aMaNz\#$
6	$za\underline{z}abzbz$	q	$MaNz\#$
7	$za\underline{z}abzbz$	q	$zaNz\#$
8	$zaz\underline{a}bzbz$	q	$aNz\#$
9	$zaza\underline{b}zbz$	q	$Nz\#$
10	$zaza\underline{b}zbz$	q	$bNbz\#$
11	$zazab\underline{z}bz$	q	$Nbz\#$
12	$zazab\underline{z}bz$	q	$zbz\#$
13	$zazabz\underline{b}z$	q	$bz\#$
14	$zazabzb\underline{z}$	q	$z\#$
15	$zazabzbz\_$	q	$\#$
16	$zazabzbz\_$	f	λ

A l'étape 4, le symbole non terminal M est au sommet de la pile. Il y a donc 2 transitions possibles, $M \to aMa$ et $M \to z$. La première a été choisie. Si la deuxième avait été choisie, la lecture de la séquence d'entrée n'aurait pu mener à l'état final. L'automate est donc non déterministe : il y a parfois un choix de transitions. Pour qu'une séquence soit acceptée, il suffit qu'il y ait une suite de transitions menant de l'état initial à l'état final, tout en faisant la lecture de toute la séquence d'entrée.

Un compilateur faisant l'analyse syntaxique d'un programme Pascal ou C pourrait procéder exactement comme l'automate à pile ci-dessus. L'une de ses structures de données serait donc une pile. □

2.28 Exercice. Trouvez une grammaire qui génère le langage $\{a^n b^n : n \in \mathbf{N}\}$, puis utilisez la technique du théorème 2.26 pour construire un automate à pile qui accepte ce langage. Donnez la suite de configurations menant à l'acceptation de la séquence ab.

□

Ainsi donc, à chaque grammaire non contextuelle correspond un automate à pile qui accepte le langage généré par la grammaire. L'inverse est aussi vrai.

2.29 Théorème. *Pour chaque automate à pile M, il existe une grammaire non contextuelle G telle que $L(M) = L(G)$.*

DÉMONSTRATION. Voir [Brookshear89]. □

Il est important de connaître ce résultat, car, en conjonction avec le théorème 2.26, il montre que les automates à pile et les grammaires non contextuelles caractérisent la même classe de langages. Cependant, vous n'aurez pas à appliquer le théorème 2.29, car, même pour un automate à pile simple, il conduit à une grammaire non contextuelle monstrueuse.

Une conséquence immédiate des théorèmes 2.26 et 2.29 est que, pour montrer qu'un langage est non contextuel, il suffit de donner une grammaire non contextuelle qui le génère ou un automate à pile qui l'accepte.

Nous allons maintenant montrer que l'union de deux langages non contextuels est un langage non contextuel. Pour y arriver, nous allons combiner deux grammaires non contextuelles de sorte que la grammaire résultante génère l'union des grammaires initiales.

2.30 Théorème. *L'union de deux langages non contextuels est un langage non contextuel.*

DÉMONSTRATION. Soient $G_1 = (V_1, T_1, S_1, R_1)$ et $G_2 = (V_2, T_2, S_2, R_2)$. Supposons que $V_1 \cap V_2 = \emptyset$ (si ce n'est pas le cas, il suffit de renommer les symboles appropriés). On cherche une grammaire G telle que $L(G) = L(G_1) \cup L(G_2)$. On définit cette grammaire $G = (V, T, S, R)$ ainsi :

$$\begin{aligned} V &= V_1 \cup V_2 \cup \{S\} \text{ où } S \notin V_1 \text{ et } S \notin V_2 \\ T &= T_1 \cup T_2 \\ R &= R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\} \end{aligned}$$

Il est facile de voir que cette grammaire génère bien l'union des langages $L(G_1)$ et $L(G_2)$. □

2.31 Exemple. Soient G_1 et G_2, les deux grammaires suivantes :

$$G_1 : \begin{array}{lcl} S_1 & \to & bS_1c \\ S_1 & \to & \lambda \end{array} \qquad G_2 : \begin{array}{lcl} S_2 & \to & abS_2 \\ S_2 & \to & \lambda \end{array}$$

Voici la grammaire G qui génère $L(G_1) \cup L(G_2)$:

$$G : \begin{array}{lcl} S & \to & S_1 \\ S & \to & S_2 \\ S_1 & \to & bS_1c \\ S_1 & \to & \lambda \\ S_2 & \to & abS_2 \\ S_2 & \to & \lambda \end{array}$$

□

Voici maintenant pourquoi il est important d'avoir $V_1 \cap V_2 = \emptyset$. Soient

$$G_1 : \begin{array}{lcl} S_1 & \to & aX \\ X & \to & b \end{array} \qquad \text{et} \qquad G_2 : \begin{array}{lcl} S_2 & \to & cX \\ X & \to & d \end{array}$$

L'union de $L(G_1)$ et $L(G_2)$ est $\{ab, cd\}$. Si on applique l'algorithme du théorème 2.30 sans renommer X, on obtient la grammaire G suivante.

$$G : \begin{array}{lcl} S & \to & S_1 \\ S & \to & S_2 \\ S_1 & \to & aX \\ X & \to & b \\ S_2 & \to & cX \\ X & \to & d \end{array}$$

Mais $L(G) = \{ab, ad, cb, cd\} \neq L(G_1) \cup L(G_2)$.

Afin d'éviter ce problème, transformons G_2 en renommant X dans G_2, de sorte que $V_1 \cap V_2 = \emptyset$. On obtient alors

$$\begin{array}{lrcl} G_2' : & S_2 & \to & cY \\ & Y & \to & d \end{array}$$

Ensuite, si on combine G_1 et G_2' en appliquant l'algorithme, on obtient G telle que $L(G) = L(G_1) \cup L(G_2')$.

$$\begin{array}{lrcl} G : & S & \to & S_1 \\ & S & \to & S_2 \\ & S_1 & \to & aX \\ & X & \to & b \\ & S_2 & \to & cY \\ & Y & \to & d \end{array}$$

2.2.1 Forme normale de Chomsky

Soit $G = (V, T, S, R)$ une grammaire non contextuelle. Si $\lambda \in L(G)$, R doit contenir au moins une règle-λ. Par contre, si $\lambda \notin L(G)$, il est possible d'éliminer toutes les règles-λ.

2.32 Théorème. *Soit $G = (V, T, S, R)$ une grammaire non contextuelle telle que $\lambda \notin L(G)$. Il existe une grammaire G' sans règle-λ telle que $L(G) = L(G')$.*

DÉMONSTRATION. L'élimination de toutes les règles-λ d'une grammaire ne générant pas la séquence vide se fait en trois étapes. Celles-ci seront illustrées au moyen de la grammaire de l'exemple 2.33 qui suit.

2.33 Exemple.

$$\begin{array}{lcl} S & \to & cPcQc \\ P & \to & aPa \\ P & \to & QN \\ Q & \to & bPb \\ Q & \to & \lambda \\ N & \to & aNb \\ N & \to & \lambda \end{array}$$

□

1. Soient
   - $U_0 = \{X \in V : \exists r \in R : r = (X \to \lambda)\}$. U_0 contient tous les non terminaux pouvant générer λ après une seule étape de dérivation.

- $U_n = U_{n-1} \cup \{X \in V : (\exists r \in R : (r = (X \to X_1 X_2 \ldots X_m) \wedge m > 1 \wedge X_i \in U_{n-1}, 1 \leq i \leq m))\}$

  Puisque V est fini, il existe un n tel que $U_n = U_{n-1}$. Appelons U le U_n tel que $U_n = U_{n-1}$. Bien sûr, $S \notin U$, car $\lambda \notin L(G)$. U contient tous les symboles non terminaux qui sont à l'origine d'une séquence de dérivation menant à la génération de λ. Pour tout $X \in U$ nous notons ce fait par $X \to^* \lambda$, nous disons que X est un symbole générateur de λ.

  Pour la grammaire de l'exemple 2.33, $U_0 = \{N, Q\}$ et $U_1 = U_0 \cup \{P\} = \{N, P, Q\}$. Comme $U_1 = U_2 = U_3 = \cdots = \{N, P, Q\}$, $U = \{N, P, Q\}$.

2. Poser $G' = G$, puis, pour chaque règle de la forme $N \to w$, ajouter à G' les règles de la forme $N \to w'$, où w' est obtenue de w en y supprimant un ou plusieurs des symboles de U (faire toutes les combinaisons). Dans le cas de la grammaire de l'exemple 2.33, il faut ajouter les règles suivantes :

S	$\to$	$ccQc$	} viennent de $S \to cPcQc$
S	$\to$	$cPcc$	
S	$\to$	ccc	
P	$\to$	aa	} vient de $P \to aPa$
P	$\to$	λ	} viennent de $P \to QN$
P	$\to$	N	
P	$\to$	Q	
Q	$\to$	bb	} vient de $Q \to bPb$
N	$\to$	ab	} vient de $N \to aNb$

3. Enlever les règles-λ ($P \to \lambda$, $Q \to \lambda$ et $N \to \lambda$ dans l'exemple). Le résultat final est la grammaire G' :

$S \to cPcQc$	$P \to aPa$	$Q \to bPb$
$S \to ccQc$	$P \to aa$	$Q \to bb$
$S \to cPcc$	$P \to QN$	$N \to aNb$
$S \to ccc$	$P \to N$	$N \to ab$
	$P \to Q$	

Avec la nouvelle grammaire G' on peut dériver les mêmes séquences terminales qu'avec la grammaire originale G (voir [Brookshear89, page 95], pour une justification plus rigoureuse). Par exemple :

Dérivation par G			Règle utilisée
S	$\Rightarrow$	$cPcQc$	$S \to cPcQc$
	$\Rightarrow$	$caPacQc$	$P \to aPa$
	$\Rightarrow$	$caQNacQc$	$P \to QN$
	$\Rightarrow$	$caNacQc$	$Q \to \lambda$
	$\Rightarrow$	$caacQc$	$N \to \lambda$
	$\Rightarrow$	$caacc$	$Q \to \lambda$

Dérivation par G'			Règle utilisée
S	$\Rightarrow$	$cPcc$	$S \to cPcc$
	$\Rightarrow$	$caacc$	$P \to aa$

□

2.34 Exercice. Soit la grammaire G suivante :

$$\begin{array}{ll} S \to aSb & N \to S \\ S \to dNa & N \to \lambda \end{array}$$

1. Quel est le langage généré par cette grammaire ?
2. Cette grammaire ne génère pas λ, bien qu'elle contienne une règle-λ. Utilisez l'algorithme présenté au théorème 2.32 pour trouver une grammaire équivalente sans règle-λ.

□

2.35 Définition. Une grammaire non contextuelle dont le côté droit de chaque règle consiste soit en un seul symbole terminal, soit en exactement deux symboles non terminaux, est dite sous la *forme normale de Chomsky.* □

Le théorème qui suit utilise le théorème 2.32 pour montrer que toute grammaire non contextuelle qui ne génère pas la séquence vide peut être transformée en une grammaire équivalente qui a la forme normale de Chomsky.

2.36 Théorème. *Si L est un langage non contextuel tel que $\lambda \notin L$, alors il existe une grammaire non contextuelle G, sous forme normale de Chomsky, et telle que $L(G) = L$.*

DÉMONSTRATION. Soit $G = (V, T, S, R)$ une grammaire non contextuelle sans règle-λ telle que $L(G) = L$ (une telle grammaire existe, par le théorème 2.32). Il faut trois étapes pour construire, à partir de G, une grammaire équivalente G' sous forme normale de Chomsky. Ces étapes seront illustrées au moyen d'un exemple.

1. Poser $G' = G$. Pour chaque $x \in T$, soit $X \notin V$ un nouveau symbole non terminal :
   • Remplacer toutes les occurrences de x par X dans chaque règle de G'.
   • Ajouter la règle $X \to x$ à G'.
   Le résultat est que le côté droit de chaque règle de la grammaire G' est
   • soit un seul terminal ;
   • soit une suite de non terminaux.
   De plus, $L(G') = L(G)$. Par exemple, soit la grammaire G :

   $$\begin{array}{ll} S \to cMc & M \to bMb \\ M \to N & N \to a \end{array}$$

   On a $\lambda \notin L(G)$, car toutes les séquences commencent par un c. Le résultat de la transformation décrite ci-dessus est G' :

   $$\begin{array}{lll} S \to CMC & N \to A & B \to b \\ M \to N & A \to a & C \to c \\ M \to BMB & & \end{array}$$

2. Dans G', remplacer chaque règle de la forme $N \to N_1 N_2 \ldots N_n$ (où $n > 2$ et N_i non terminal) par l'ensemble des règles

   $$\begin{array}{lcl} N & \to & N_1 R_1 \\ R_1 & \to & N_2 R_2 \\ & \vdots & \\ R_{n-3} & \to & N_{n-2} R_{n-2} \\ R_{n-2} & \to & N_{n-1} N_n \end{array}$$

   Pour notre exemple, le résultat est G' :

   $$\begin{array}{lll} S \to CR_1 & M \to BP_1 & A \to a \\ R_1 \to MC & P_1 \to MB & B \to b \\ M \to N & N \to A & C \to c \end{array}$$

3. G' a maintenant des règles de la forme

   $$\begin{array}{lcll} \textit{non terminal} & \to & \textit{terminal} & \\ \textit{non terminal} & \to & \textit{non terminal} & \textit{non terminal} \\ \textit{non terminal} & \to & \textit{non terminal} & \end{array}$$

   Il reste maintenant à éliminer celles du type *non terminal* $\to$ *non terminal*. Pour cela, il faut considérer toutes les chaînes $N_n \to N_{n-1} \to N_{n-2} \to \cdots \to N_2 \to N_1$ et ajouter la règle
   • $N_n \to x$ si $N_1 \to x$ est une règle de G',
   • $N_n \to AB$ si $N_1 \to AB$ est une règle de G'.
   Dans notre exemple, les chaînes de G' qui nous intéressent sont les trois suivantes :

(a) $N \to A$;

(b) $M \to N$;

(c) $M \to N$, $N \to A$.

On ajoute
- $N \to a$ car $A \to a$ est une règle de G'.
- $M \to a$ car $A \to a$ dans G'.

(Si on avait une règle $N \to b$ dans G', on ajouterait $M \to b$.) On enlève ensuite toutes les productions de la forme *non terminal* $\to$ *non terminal*. La grammaire finale résultante est :

$S \to CR_1$	$M \to BP_1$	$A \to a$
$R_1 \to MC$	$P_1 \to MB$	$B \to b$
$M \to a$	$N \to a$	$C \to c$

□

Ce résultat est intéressant pour au moins deux raisons. Tout d'abord, ce type de grammaire a une structure très simple ; les preuves visant à démontrer certaines propriétés des langages non contextuels peuvent être plus simples si elles supposent qu'une grammaire non contextuelle a la forme normale de Chomsky (à la condition qu'elle ne génère pas λ). Ensuite, il est surprenant de constater que la syntaxe d'un langage de programmation comme C ou Java pourrait être décrite au moyen d'une grammaire dont la structure est aussi simple. Il faut cependant noter que cette simplicité structurelle résulte en un inconvénient majeur : il faut un plus grand nombre de règles pour définir un langage.

2.37 Exercice. Transformez la grammaire obtenue à l'exercice 2.34 en grammaire sous forme normale de Chomsky.

□

Que faire si $\lambda \in L$? Si L est non contextuel, on peut trouver une grammaire qui est « presque » sous la forme normale de Chomsky :

1. D'abord trouver $G = (V, T, S, R)$ sous forme normale de Chomsky et telle que $L(G) = L - \{\lambda\}$.
2. Ajouter un nouveau non terminal S' et la règle $S' \to \lambda$ (la nouvelle grammaire génère donc λ). S' est le nouveau symbole initial.
3. Pour chaque règle de R de la forme $S \to w$, ajouter une règle $S' \to w$.

2.38 Exemple. Supposons que $\lambda \in L$ et que $L - \{\lambda\}$ est le langage généré par la grammaire

$$S \to MN \qquad N \to MS \qquad N \to a \qquad M \to a$$

Comme cette grammaire est déjà sous la forme normale de Chomsky, il suffit d'exécuter les deux dernières étapes. S' est le nouveau symbole initial.

$$\begin{array}{lll} S' \to \lambda & S \to MN & N \to a \\ S' \to MN & N \to MS & M \to a \end{array}$$

□

2.2.2 Problèmes

1. Construisez une grammaire non contextuelle qui génère le langage

   $$\{a^r b^s c^t : r \in \mathbf{N} \wedge s \in \mathbf{N} \wedge t \in \mathbf{N} \wedge s = r + t\}.$$

2. Convertissez la grammaire suivante, dont le symbole initial est S, en grammaire sous forme normale de Chomsky.

   $$S \to McN \qquad M \to aM \qquad N \to bN$$
   $$M \to \lambda \qquad N \to \lambda$$

3. Soit $\Sigma = \{a, b, \circ, \cup, *,), (, \emptyset\}$. Construisez un automate à pile M tel que $L(M)$ est l'ensemble de toutes les séquences de Σ^* qui sont des expressions régulières sur l'alphabet $\{a, b\}$. Suggestion : Il est plus facile de trouver une grammaire non contextuelle qui génère le langage demandé que de construire l'automate ; une approche simple consiste donc à trouver la dite grammaire, puis à appliquer le théorème 2.26.

4. Donnez une grammaire non contextuelle qui génère le langage sur l'alphabet $\{a, b\}$ dont les éléments sont les séquences qui contiennent le même nombre de a que de b.

2.3 Les limites des automates à pile

OBJECTIFS SPÉCIFIQUES

1. Classifier des automates à pile décrits par des diagrammes de transitions. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix.
2. Classifier des descriptions formelles d'automates à pile. Dire tout d'abord si la description est correcte. Si oui, dire si l'automate est déterministe. Expliquer la raison de chaque choix.
3. Construire un automate à pile déterministe acceptant un langage donné.
4. Démontrer qu'un langage donné n'est pas un langage non contextuel.

2.3.1 La portée des langages non contextuels

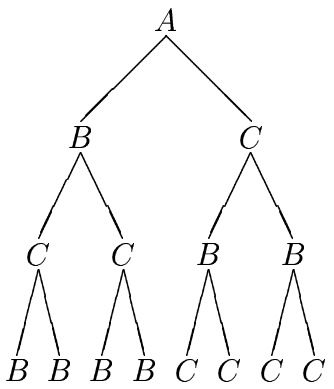
Il existe des langages qui ne sont pas non contextuels (on dit qu'ils sont *contextuels* ou *dépendants du contexte*). Au chapitre précédent, nous avons présenté une méthode permettant de montrer qu'un langage n'est pas régulier. Nous verrons maintenant comment montrer qu'un langage est contextuel.

Le théorème 2.40 est appelé *lemme de pompage*, car il montre comment créer de nouvelles séquences en « pompant » des sous-séquences, c'est-à-dire en ajoutant un nombre quelconque de ces sous-séquences. C'est ce théorème qui sera utilisé pour montrer qu'un langage est contextuel.

Dans la preuve du théorème 2.40 nous allons utiliser le lemme suivant.

2.39 Lemme.
Dans un arbre binaire complet de hauteur k, le nombre de feuilles est au plus 2^k. □

Nous avons $m = 2$ est le nombre de symboles du côté droit d'une règle à part ceux génénérant un terminal. Chaque noeud d'un arbre de dérivation a au plus m enfants ; donc, un arbre de dérivation de profondeur k peut produire une séquence de longueur au plus $m^k = 2^k$). Ainsi, l'arbre suivant a une profondeur 3 et produit une séquence de longueur maximale, soit 8.



longueur $\leq 2^0 = 1$ (profondeur 0)

longueur $\leq 2^1 = 2$ (profondeur 1)

longueur $\leq 2^2 = 4$ (profondeur 2)

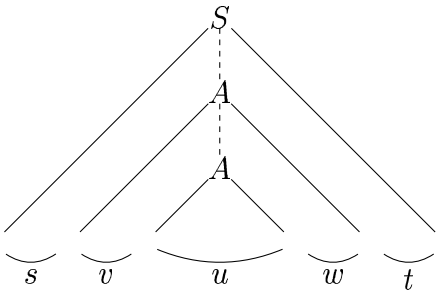
longueur $\leq 2^3 = 8$ (profondeur 3)

2.40 Théorème.
Soit L un langage infini. Si L est non contextuel, alors il existe un nombre $p \in \mathbf{N}^+$, dépendant de L et d'une grammaire sous forme normale de Chomsky le générant, tel que la propriété suivante est vraie. $\forall z \in L : \exists s, v, u, w, t \in \Sigma^* : z = svuwt$ et $|z| \geq p$ et $|vuw| \leq p$ et $|vw| \geq 1$ $(v \neq \lambda \vee w \neq \lambda)$ alors $\forall i \in \mathbf{N} : sv^iuw^it \in L$.

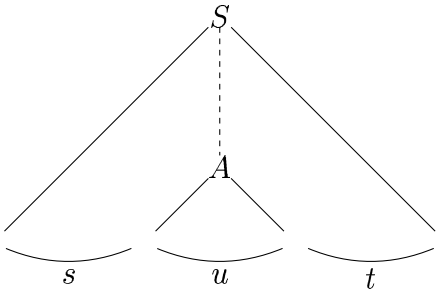
DÉMONSTRATION. Nous supposons que L est engendré par une grammaire sous forme normale de Chomsky et que le nombre de symboles non terminaux de cette grammaire est k. Nous considérons les langages ne contenant pas la séquence vide. Pour un langage qui contient la séquence vide, comme cette dernière ne nous intéresse pas pour le choix de z, nous pouvons considérer ce langage privé de la séquence vide. Nous allons montrer que $p = 2^k$ est la constante désirée du langage.

Pour cela, soit z un mot de L. Considérons son arbre de dérivation. Comme la grammaire est sous forme normale de Chomsky, cet arbre est un arbre binaire complet, sauf pour les feuilles qui sont seuls fils de nœuds. Si nous négligeons les feuilles, il

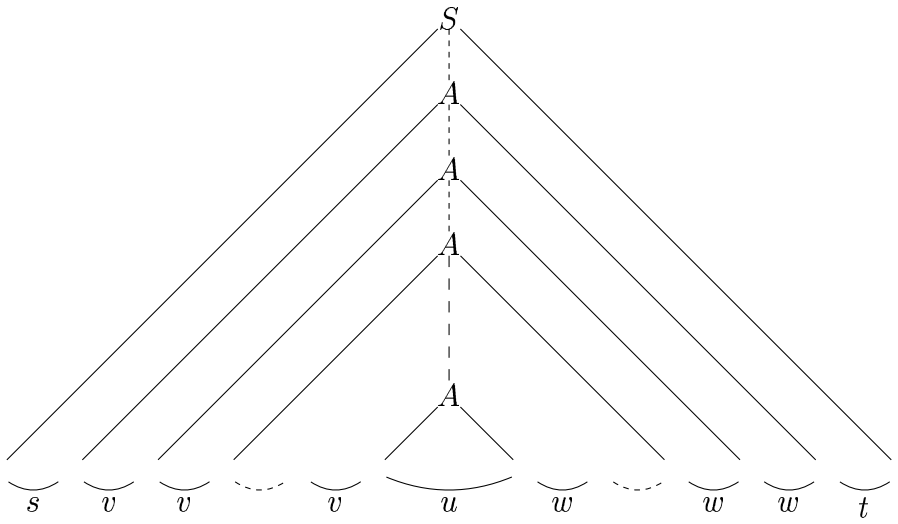
reste un arbre binaire complet dont le nombre des nœuds externes (les feuilles) est exactement le même que la longueur de z. Si la longueur de z est au moins 2^k, la hauteur de l'arbre est au moins k. Notez qu'un chemin de longueur k possède k arêtes et $k+1$ nœuds. Sur un chemin de longueur k étiqueté par des noms de variables, un même nom doit apparaître deux fois (principe des nids de pigeons). Considérons l'ensemble I des nœuds i tels que la variable de i apparaît encore une fois sur un chemin partant de i vers une feuille de l'arbre de dérivation. L'ensemble I contient au moins un élément. Parmi les nœuds de I, choisissons celui qui est à la racine du sous-arbre ayant la plus petite hauteur, soit n ce nœud et A son étiquette. Nous sommes sûrs que le sous-arbre issu de n est de hauteur au plus k, sinon il y aurait un autre sous-arbre de hauteur plus petite dont le nœud est répété plus d'une fois. Par conséquent, le nombre de feuilles de ce dernier est au plus $p = 2^k$. La situation est la suivante :



Par conséquent $|vuw| \leq p = 2^k$. Puisque la grammaire est sous forme normale de Chomsky, chaque nœud, à part celui juste avant une feuille, a exactement deux fils. Donc, v et w ne sont pas simultanément vide. Dans la figure ci-dessus le plus petit sous arbre peut-être mis à la place du plus grand produisant le mot sut.



Le plus grand peut remplacer le plus petit autant de fois qu'on veut, produisant sv^iuw^it, $i \in \mathbf{N}$, comme le montre le schéma qui suit.



□

2.41 Exemple. Soit la grammaire, sous forme normale de Chomsky, $G = (\{A, B, C\}, \{a, b\}, A, R)$, où R est

$A \to BC$ $\quad$ $B \to a$ $\quad$ $C \to BB$
$B \to CC$ $\quad$ $C \to b$ $\quad$ $C \to BC$

Le langage généré par G est infini. En effet, toutes les séquences de la forme $a^{n+1}b$, où $n \in \mathbf{N}$, appartiennent à $L(G)$. En effet, on a la dérivation suivante de $a^{n+1}b$.

A
$\Rightarrow$ $\quad$ ⟨ Règle $A \to BC$. ⟩
BC
$\Rightarrow$ $\quad$ ⟨ Règle $C \to BC$. ⟩
BBC
$\Rightarrow$ $\quad$ ⟨ Règle $B \to a$. ⟩
aBC
$\Rightarrow$ $\quad$ ⟨ Les deux étapes précédentes peuvent être faites zéro, une ou plusieurs fois. ⟩
...

$\Rightarrow$ $\langle$ À la ligne suivante, $n \in \mathbf{N}$. $\rangle$

$a^n BC$

$\Rightarrow$ $\langle$ Règle $B \to a$. $\rangle$

$a^{n+1} C$

$\Rightarrow$ $\langle$ Règle $C \to b$. $\rangle$

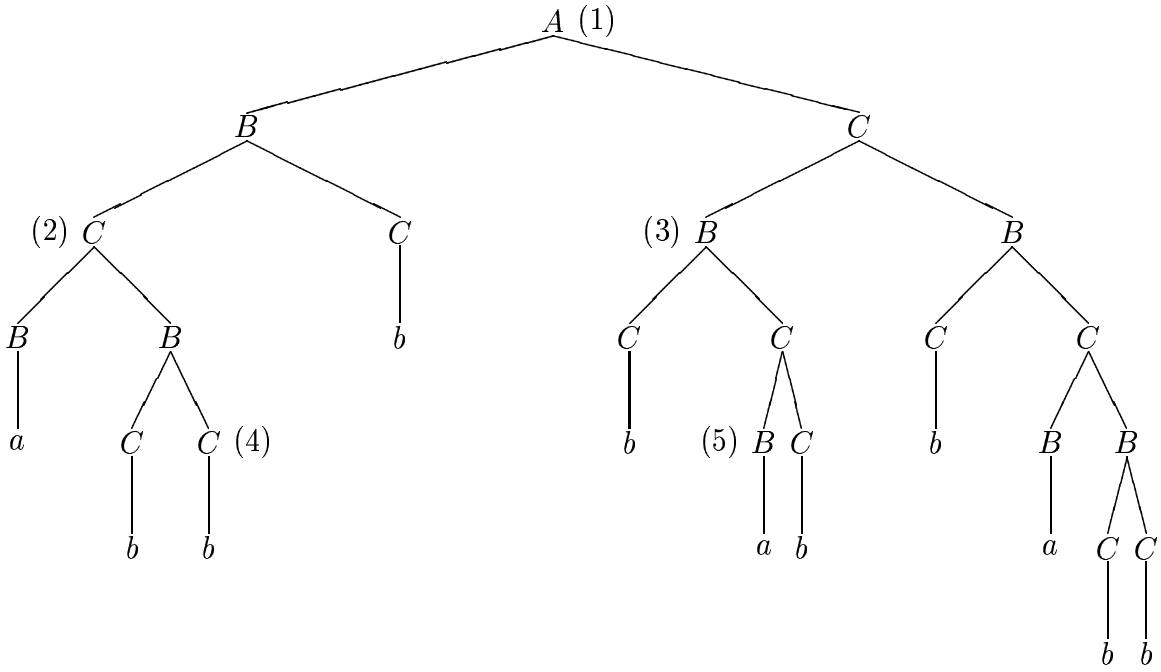
$a^{n+1} b$.

Soit j = nombre de symboles non terminaux = $|V|$. Pour notre exemple, $j = 3$. Choisissons une séquence $z \in L$ telle que $|z| > 2^j$ (une telle séquence existe, puisque L est infini). Pour notre exemple, $|z| > 2^3 = 8$. Prenons la séquence

$z = abbbbabbabb$

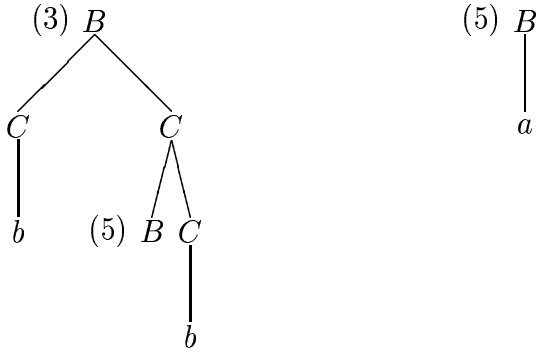
(on a $|z| = 11$). N'importe quel arbre de dérivation de z a une profondeur plus grande que $j = 3$. Il y a donc au moins un chemin, de la racine à une feuille, contenant plus de j symboles non terminaux.

Voici l'arbre de dérivation de z.



Sur cet arbre les nœuds (2) et (3) correspondent à deux sous-arbres de hauteur minimale et tel que la variable étiquetant le nœud est répétée deux fois.

Prenons, le chemin $ACBCB$. Il est de profondeur 4 et contient 5 symboles non terminaux. Considérons le sous-arbre dont la racine est B (3) duquel on enlève le sous-arbre dont la racine est B (5) (on garde la racine).



La séquence z peut être décomposée de la manière suivante :

$$\underbrace{abbb}_{s}\underbrace{b}_{v}\underbrace{a}_{u}\underbrace{b}_{w}\underbrace{babb}_{t}$$

où

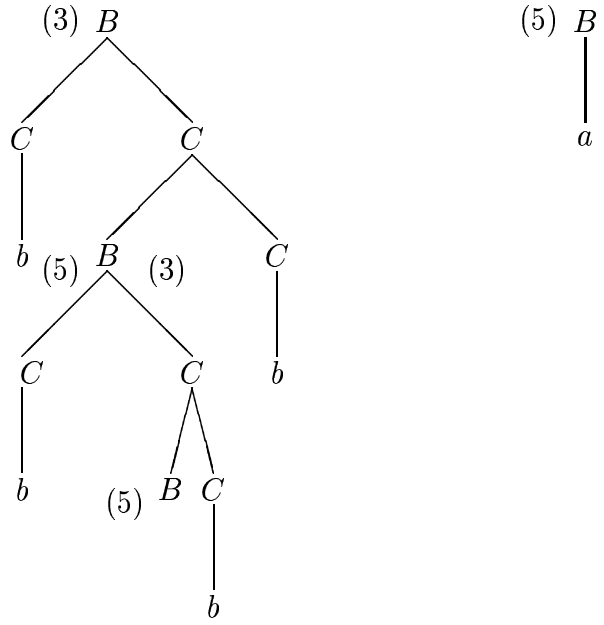
- s est la séquence terminale à gauche du sous-arbre (3) dans l'arbre principal (1) ;
- v est la séquence terminale à gauche du sous-arbre (5) dans le sous-arbre (3) ;
- u est la séquence terminale du sous-arbre (5) ;
- w est la séquence terminale à droite du sous-arbre (5) dans le sous-arbre (3) ;
- t est la séquence terminale à droite du sous-arbre (3) dans l'arbre principal (1).

Nous avons bien $|vuw| = |bab| = 3 \leq 2^3 = 8$ et $|vw| \geq 1$ puisque v et w sont toutes deux non vides.

Le sous-arbre (3) correspond à la dérivation

$$B \Rightarrow CC \Rightarrow bC \Rightarrow bBC \Rightarrow bBb$$

On peut appliquer au B de la séquence résultante bBb la même suite de dérivations, ce qui donne la séquence $bbBbb$ (début du « pompage »). Cela correspond à greffer tout le sous-arbre (3) à la position (5) :



On peut appliquer la suite de dérivations ci-dessus au B de $bbBbb$ autant de fois que l'on veut (ce qui revient à greffer le sous-arbre (3) en position (5) autant de fois que désiré). Finalement, on peut utiliser la production $B \to a$, ce qui donne la séquence $b^n a b^n$, où $n \in \mathbf{N}^+$ (ce qui revient à replacer le sous-arbre (5) à la position (5)). Après toutes ces étapes, la séquence obtenue est

$$\underbrace{abbb}_{s}\underbrace{b^n}_{v^n}\underbrace{a}_{u}\underbrace{b^n}_{w^n}\underbrace{babb}_{t}$$

On voit comment on réussit à répéter les sous-séquences v et w autant de fois qu'on le désire. Cela nous permet d'obtenir, à partir de la séquence $z = abbbbabbabb$, de nouvelles séquences $abbbb^n ab^n babb$, pour $n \in \mathbf{N}$, qui appartiennent à $L(G)$, tel qu'annoncé dans l'énoncé du théorème. Le cas $n = 0$ est obtenu en greffant le sous-arbre (5) à la palce du sous-arbre (3).

Considérons maintenant le sous-arbre de racine (2) et le chemin $ABCBC$ le plus à droite. Ce chemin contient deux répétitions de C, l'une en (2) et l'autre en (4).

La séquence $abbbbabbabb$ peut maintenant être divisée sous la forme

$$\underbrace{}_{s}\underbrace{ab}_{v}\underbrace{b}_{u}\underbrace{}_{w}\underbrace{bbabbabb}_{t}$$

où

- s est la séquence terminale à gauche du sous-arbre (2) dans l'arbre principal (1) (c'est la séquence vide) ;
- v est la séquence terminale à gauche du sous-arbre (4) dans le sous-arbre (2) ;
- u est la séquence terminale du sous-arbre (4) ;
- w est la séquence terminale à droite du sous-arbre (4) dans le sous-arbre (2) (c'est la séquence vide) ;
- t est la séquence terminale à droite du sous-arbre (2) dans l'arbre principal (1).

Puisque nous avons $|vuw| = |abb| = 3 \leq 2^3 = 8$ et $|vw| = |ab| \geq 1$. Le théorème nous indique que les séquences

$$\underbrace{}_{s}\underbrace{(ab)^n}_{v^n}\underbrace{b}_{u}\underbrace{}_{w^n}\underbrace{bbabbabb}_{t}$$

où $n \in \mathbf{N}$ appartiennent au langage généré par G. Cela est facile à vérifier en utilisant les mêmes arguments que ci-dessus.

□

Grâce à ce théorème, on peut montrer que certains langages dépendent du contexte. Toutefois, afin d'en permettre une utilisation plus facile, nous allons l'énoncer de manière différente dans le corollaire suivant.

2.42 Corollaire. *Soit L un langage infini. Si, $\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s, v, u, w, t \in \Sigma^* : (z = svuwt \wedge |z| \geq p \wedge |vuw| \leq p \wedge |vw| \geq 1) \wedge (\exists i \in \mathbf{N} : sv^iuw^it \notin L)$ alors L n'est pas un langage non contextuel (est contextuel).*

DÉMONSTRATION. Nous allons montrer que la proposition

> Si L est un langage infini non contextuel, alors il existe un nombre $p \in \mathbf{N}^+$, dépendant de L et d'une grammaire sous forme normale de Chomsky le générant, tel que la propriété suivante est vraie. $\forall z \in L : \exists s, v, u, w, t \in \Sigma^* : z = svuwt$ et $|z| \geq p$ et $|vuw| \leq p$ et $|vw| \geq 1$ $(v \neq \lambda \vee w \neq \lambda)$ alors $\forall i \in \mathbf{N} : sv^iuw^it \in L$.

tirée du théorème 2.40, est équivalente à la proposition

> Si, $\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s, v, u, t, w \in \Sigma^* : z = svuwt$ et $|z| \geq p \wedge |vuw| \leq p \wedge |vw| \geq 1) \wedge (\exists i \in \mathbf{N} : sv^iuw^it \notin L)$ alors L n'est pas un langage non contextuel (est contextuel).

du présent corollaire. Puisque le théorème a été démontré, cela démontrera le corollaire.

Supposons que L est un langage infini sur l'alphabet Σ.

> Si L est un langage infini non contextuel, alors il existe un nombre $p \in \mathbf{N}^+$, dépendant de L et d'une grammaire sous forme normale de Chomsky le générant, tel que la propriété suivante est vraie. $\forall z \in L : \exists s, v, u, w, t \in \Sigma^* : z = svuwt$ et $|z| \geq p$ et $|vuw| \leq p$ et $|vw| \geq 1$ $(v \neq \lambda \vee w \neq \lambda)$ alors $\forall i \in \mathbf{N} : sv^iuw^it \in L$.

$\Leftrightarrow$ $\langle$ Reformulation symbolique. $\rangle$

L non contextuel $\Rightarrow (\exists p \in \mathbf{N}^+ : \forall z \in L : \exists s,v,u,w,t \in \Sigma^* : z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1 \Rightarrow (\forall i \in \mathbf{N} : sv^iuw^it \in L))$

$\Leftrightarrow$ $\langle$ $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$ (faire la table de vérité). $\rangle$

$\neg\ (\exists p \in \mathbf{N}^+ : \forall z \in L : \exists s,v,u,w,t \in \Sigma^* : z = suvwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1 \Rightarrow (\forall i \in \mathbf{N} : sv^iuw^it \in L))$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ $\neg(\exists x : p(x)) \Leftrightarrow (\forall x : \neg p(x))$. $\rangle$

$\forall p \in \mathbf{N}^+ : \neg\ (\forall z \in L : \exists s,v,u,w,t \in \Sigma^* : z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1 \Rightarrow (\forall i \in \mathbf{N} : sv^iuw^it \in L))$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ $\neg(\forall x : p(x)) \Leftrightarrow (\exists x : \neg p(x))$. $\rangle$

$\forall p \in \mathbf{N}^+ : \exists z \in L : \neg\ (\exists s,v,u,w,t \in \Sigma^* : z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1 \Rightarrow (\forall i \in \mathbf{N} : sv^iuw^it \in L))$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ $\neg(\exists x : p(x)) \Leftrightarrow (\forall x : \neg p(x))$. $\rangle$

$\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s,v,u,w,t \in \Sigma^* : \neg\ (z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1 \Rightarrow (\forall i \in \mathbf{N} : sv^iuw^it \in L))$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ $\neg(p \Rightarrow q) \Leftrightarrow \neg(\neg p \vee q) \Leftrightarrow (p \wedge \neg q)$ (table de vérité de $\Rightarrow$ et loi de de Morgan). $\rangle$

$\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s,v,u,w,t \in \Sigma^* : (\ z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1) \wedge \neg(\forall i \in \mathbf{N} : sv^iuw^it \in L)$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ $\neg(\forall x : p(x)) \Leftrightarrow (\exists x : \neg p(x))$. $\rangle$

$\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s,v,u,w,t \in \Sigma^* : (z = svuwt \wedge |z| \geq p \wedge$
$|vuw| \leq p \ \wedge |vw| \geq 1) \wedge (\exists i \in \mathbf{N} : sv^iuw^it \notin L)$
$\Rightarrow \neg$ (L non contextuel)

$\Leftrightarrow$ $\langle$ Reformulation en français. $\rangle$

Si, $\forall p \in \mathbf{N}^+ : \exists z \in L : \forall s,v,u,t,w \in \Sigma^* : z = svuwt$ et $|z| \geq p \ \wedge |vuw| \leq p \ \wedge |vw| \geq 1 \ \wedge \ (\exists i \in \mathbf{N} : sv^iuw^it \notin L)$ alors L n'est pas un langage non contextuel (est contextuel).

□

Voyons maintenant comment utiliser ce corollaire pour montrer qu'un langage dépend du contexte.

2.43 Exemple. Montrons que le langage $L = \{a^k b^k c^k : k \in \mathbf{N}\}$ est contextuel. C'est bien un langage infini ; nous pouvons donc appliquer le corollaire 2.42.

Soit $p \in \mathbf{N}^+$ un nombre quelconque, considérons une séquence quelconque $a^k b^k c^k \in L$ où $k \geq p$ et supposons qu'elle est décomposée sous la forme $svuwt$, où au moins l'une de v ou w est non vide et $|vuw| \leq p$. Il faut montrer qu'il existe un $i \in \mathbf{N}$ tel que $sv^i uw^i t \notin L$. Puisque nous ne connaissons par la forme exacte des sous-séquences s, t, u, v, w, il nous faudra étudier tous les cas possibles.

1. Si v contient des a et des b, la séquence sv^2uw^2t contient un b qui précède un a (rappel : $(ab)^2 = abab$). Ce n'est donc pas une séquence de L. Il en est de même si v contient à la fois des a et des c, ou des b et des c. Notez que la séquence vuw a une longueur plus petite ou égale à k, par conséquent, elle ne peut introduire qu'au plus deux symboles différents.
2. En appliquant le même raisonnement à w, on voit que si w contient au moins deux symboles différents, alors $sv^2uw^2t \notin L$.
3. Si v contient seulement des a, la séquence sv^2uw^2t contient plus de a que la séquence $svuwt$. La sous-séquence w^2 ne peut introduire que des b les c vont se trouver en déficit et $sv^2uw^2t \notin L$. Pour la même raison, si v contient seulement des b ou seulement des c, on a $sv^2uw^2t \notin L$.
4. En appliquant le même raisonnement à w, on voit que si w contient seulement des a, ou seulement des b, ou seulement des c, alors $sv^2uw^2t \notin L$.
5. Comme l'une des sous-séquences v ou w doit être non vide, nous avons épuisé toutes les possibilités.

Le langage L n'est donc pas un langage non contextuel. □

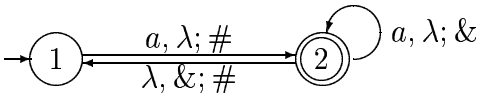
2.44 Exercice. Montrez que le langage $L = \{a^j b^k a^j b^k a^j b^k : j, k \in \mathbf{N}^+\}$ n'est pas un langage non contextuel.

□

2.3.2 Automates à pile déterministes

Dans cette sous-section, nous donnerons une définition de ce que sont les automates à pile déterministes, puis nous montrerons que ceux-ci sont moins puissants que les automates à pile non déterministes, tout en étant quand même plus puissants que les automates finis.

Les automates à pile que nous avons rencontrés jusqu'ici ne sont pas nécessairement déterministes. Par exemple, considérons l'automate suivant, dont l'alphabet de ruban est $\Sigma = \{a, b\}$ et l'alphabet de pile $\Gamma = \{\#, \&\}$.

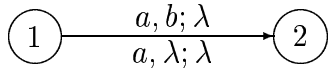


- Il n'est pas totalement défini car,
  - dans l'état 1, si le symbole d'entrée est b, aucune transition n'est possible ;
  - dans l'état 2, si b est le symbole d'entrée et $\#$ le symbole sur la pile, aucune transition n'est possible.
- Il est ambigu car, dans l'état 2, deux transitions sont possibles lorsque a est le symbole d'entrée et $\&$ le symbole sur la pile.

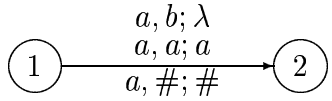
Supposons qu'on veuille construire un sous-automate déterministe qui exécute la transition

- $(1, a, b; 2, \lambda)$ si b est sur la pile
- $(1, a, \lambda; 2, \lambda)$ sinon.

Le sous-automate suivant ne fait pas l'affaire car il n'est pas déterministe (il peut choisir parmi les deux transitions si b est sur la pile).



Supposons que $\Gamma = \{a, b, \#\}$. Alors le sous-automate suivant fait ce qu'on veut et est déterministe (du moins en ce qui concerne la lecture d'un a dans l'état 1).



En effet, s'il y a un b sur la pile, il le dépile. S'il y a un autre symbole, il le dépile et le rempile, ce qui revient à ne pas toucher à la pile. Cependant il ne faut pas que la pile soit vide, car alors l'effet de la transition $a, \lambda; \lambda$ du premier sous-automate ne peut être obtenu. Une façon de s'assurer que la pile n'est pas vide est d'empiler un $\#$ au début de l'exécution et de le dépiler à la fin seulement.

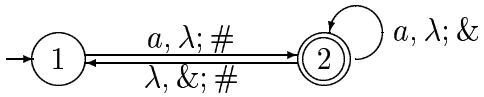
2.45 Définition. Un automate à pile $(S, \Sigma, \Gamma, T, \iota, F)$ est déterministe si et seulement si, pour tout triplet $(p, x, y) \in S \times \Sigma \times \Gamma$, il y a une et une seule transition dans T de la forme $(p, u, v; q, z)$, où

- $(u, v) \in \{(x, y), (x, \lambda), (\lambda, y), (\lambda, \lambda)\}$;
- $q \in S$;
- $z \in \Gamma$.

□

En d'autres termes, un automate à pile est déterministe si et seulement s'il y a une et une seule transition possible pour chaque triplet (état, symbole d'entrée, symbole de pile).

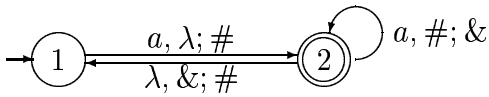
2.46 Exemple. Soit l'automate



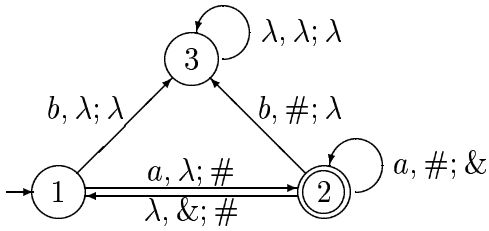
dont l'alphabet de ruban est $\Sigma = \{a, b\}$ et l'alphabet de pile $\Gamma = \{\#, \&\}$. Il n'est pas déterministe pour plusieurs raisons. En voici deux.

- Prenons $(1, b, \#) \in S \times \Sigma \times \Gamma$. Il n'y a aucune transition de la forme $(1, u, v; q, z)$ avec $(u, v) \in \{(b, \#), (b, \lambda), (\lambda, \#), (\lambda, \lambda)\}$. Autrement dit, il n'y a pas de transition à partir de l'état 1 lorsque le symbole d'entrée est un b et le symbole sur la pile un $\#$. L'automate n'est pas complètement défini.
- Prenons $(2, a, \&) \in S \times \Sigma \times \Gamma$. Il y a deux transitions de la forme $(2, u, v; q, z)$ avec $(u, v) \in \{(a, \&), (a, \lambda), (\lambda, \&), (\lambda, \lambda)\}$, soit $(2, a, \lambda; 2, \&)$ et $(2, \lambda, \&; 1, \#)$. L'automate est ambigu.

L'automate suivant, dont l'alphabet de ruban est $\Sigma = \{a, b\}$ et l'alphabet de pile $\Gamma = \{\#, \&\}$, n'est pas déterministe, car il n'est pas complètement défini. Cependant, il n'est pas ambigu.



Ajoutons un état poubelle et les transitions nécessaires afin de le rendre déterministe, sans changer le langage accepté.



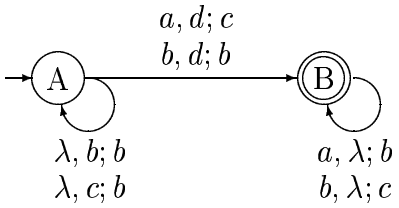
Les transitions ajoutées ne changent pas le langage accepté. Toutes ces transitions n'empilent rien, mais elles pourraient aussi empiler quelque chose sans changer le langage accepté. Cet automate est déterministe. Afin de le vérifier, montrons qu'il y a une et une seule transition pour chaque triplet (état, symbole d'entrée, symbole de pile).

État	Entrée	Pile	Transition
1	a	#	$(1, a, \lambda; 2, \#)$
1	a	&	$(1, a, \lambda; 2, \#)$
1	b	#	$(1, b, \lambda; 3, \lambda)$
1	b	&	$(1, b, \lambda; 3, \lambda)$
2	a	#	$(2, a, \#; 2, \&)$
2	a	&	$(2, \lambda, \&; 1, \#)$
2	b	#	$(2, b, \#; 3, \lambda)$
2	b	&	$(2, \lambda, \&; 1, \#)$
3	a	#	$(3, \lambda, \lambda; 3, \lambda)$
3	a	&	$(3, \lambda, \lambda; 3, \lambda)$
3	b	#	$(3, \lambda, \lambda; 3, \lambda)$
3	b	&	$(3, \lambda, \lambda; 3, \lambda)$

□

2.47 Exercice. Pour chacun des automates à pile qui suivent, dites s'il est déterministe et expliquez pourquoi. Dans le cas où un automate n'est pas déterministe, donnez toutes les situations où le non déterminisme se manifeste.

1. $M_1 = (\{A, B\}, \{a, b\}, \{b, c, d\}, T_1, A, \{B\})$ où l'ensemble de transitions T_1 est décrit par le diagramme de transitions qui suit.

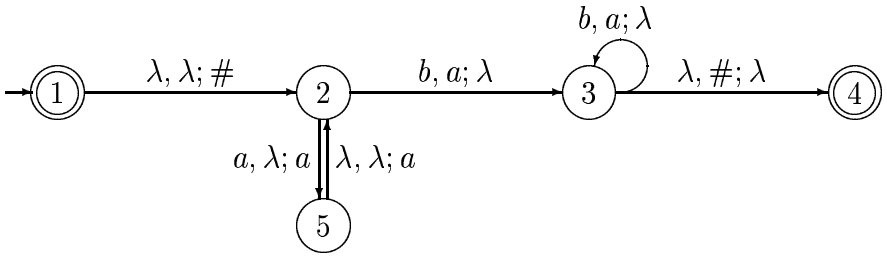


2. $M_2 = (\{A, B, C\}, \{a, b\}, \{a, b\}, T_2, A, \{A, B\})$ avec

$$T_2 = \{(A, b, \lambda; A, b), (A, a, b; B, a), (B, a, \lambda; A, a),$$
$$(B, \lambda, a; C, \lambda), (C, a, b; C, a), (C, \lambda, \lambda; B, \lambda)\}.$$

□

2.48 Exercice. Soit M un automate à pile dont l'alphabet de ruban est $\{a, b\}$, l'alphabet de pile $\{a, \#\}$ et le diagramme de transitions



Donnez un automate déterministe qui accepte le même langage. Dites quel est le langage accepté par cet automate.

□

2.49 Exercice. Dessinez la partie appropriée d'un diagramme de transitions d'un automate à pile déterministe qui passe de l'état 1 à l'état 2 (en une ou plusieurs étapes) tout en lisant un a, en empilant un c et

$$\begin{cases} \text{en dépilant un } b \text{ si } b \text{ est sur la pile,} \\ \text{en ne dépilant rien dans le cas contraire.} \end{cases}$$

Si le symbole d'entrée n'est pas un a, l'automate passe de l'état 1 à l'état poubelle 3 en lisant le b. Dans l'état 2, l'automate finit de lire la séquence d'entrée, sans toucher à la pile. Supposez que $\Sigma = \{a, b\}$, $\Gamma = \{a, b, c\}$ et que la pile n'est pas vide dans l'état 1.

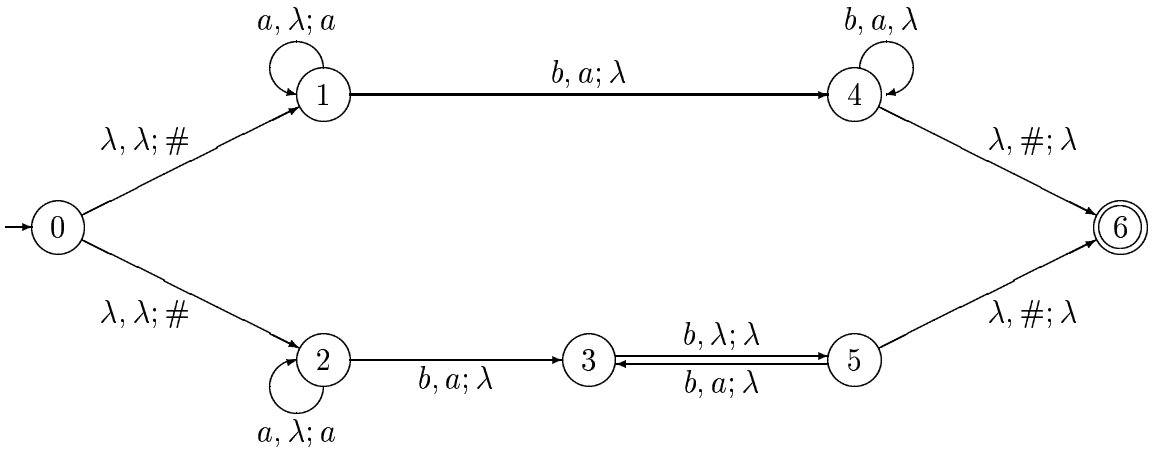
□

Montrons maintenant que les automates à pile déterministes sont moins puissants que les automates à pile non déterministes, en ce sens que la classe des langages qu'ils reconnaissent est strictement plus petite.

2.50 Théorème. *Il existe un langage non contextuel qu'aucun automate à pile déterministe ne peut accepter.*

DÉMONSTRATION. Montrons que $L = \{a^n b^n : n \in \mathbf{N}^+\} \cup \{a^n b^{2n} : n \in \mathbf{N}^+\}$ est non contextuel, mais ne peut être accepté par un automate à pile déterministe.

Premièrement, L est non contextuel, car il est accepté par l'automate à pile suivant :



Deuxièmement, montrons qu'il n'y a pas d'automate à pile déterministe qui accepte L. Procédons par contradiction. La structure de la preuve est la suivante.

il existe un automate à pile déterministe M tel que
$L(M) = \{a^n b^n : n \in \mathbf{N}^+\} \cup \{a^n b^{2n} : n \in \mathbf{N}^+\}$
$\Rightarrow$ $\langle$ À partir de M, on peut construire un automate à pile M' tel que $L(M') = \{a^n b^n c^n : n \in \mathbf{N}^+\}$. Cela sera montré à la suite de cette dérivation. $\rangle$
il existe un automate à pile M' tel que $L(M') = \{a^n b^n c^n : n \in \mathbf{N}^+\}$
$\Rightarrow$ $\langle$ Selon l'exemple 2.43, le langage $\{a^n b^n c^n : n \in \mathbf{N}^+\}$ n'est pas non contextuel ; par conséquent, il n'existe pas d'automate à pile pouvant l'accepter. $\rangle$
(il existe un automate à pile M' tel que $L(M') = \{a^n b^n c^n : n \in \mathbf{N}^+\}$) et
(il n'existe pas d'automate à pile M' tel que $L(M') = \{a^n b^n c^n : n \in \mathbf{N}^+\}$)
$\Rightarrow$ $\langle$ Contradiction. $\rangle$
faux.

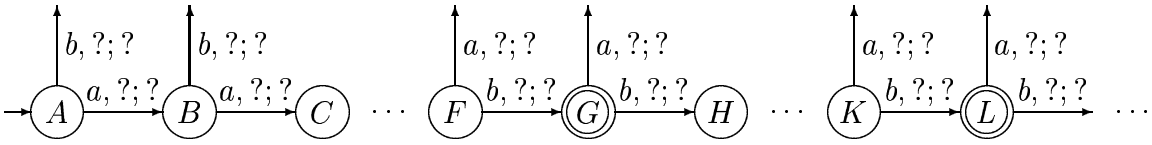
Complétons la partie manquante, c'est-à-dire montrons que si l'on dispose d'un automate à pile déterministe M tel que

$$L(M) = \{a^n b^n : n \in \mathbf{N}^+\} \cup \{a^n b^{2n} : n \in \mathbf{N}^+\},$$

alors nous pouvons construire un automate à pile M' tel que

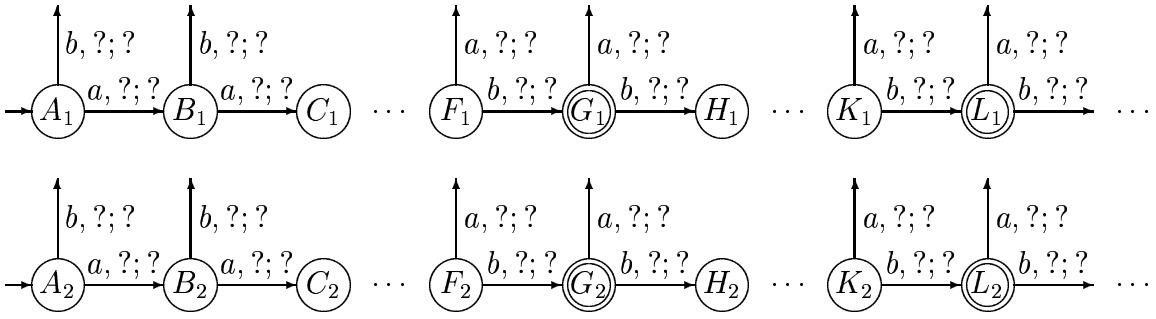
$$L(M') = \{a^n b^n c^n : n \in \mathbf{N}^+\}.$$

Imaginons une partie de l'automate M :

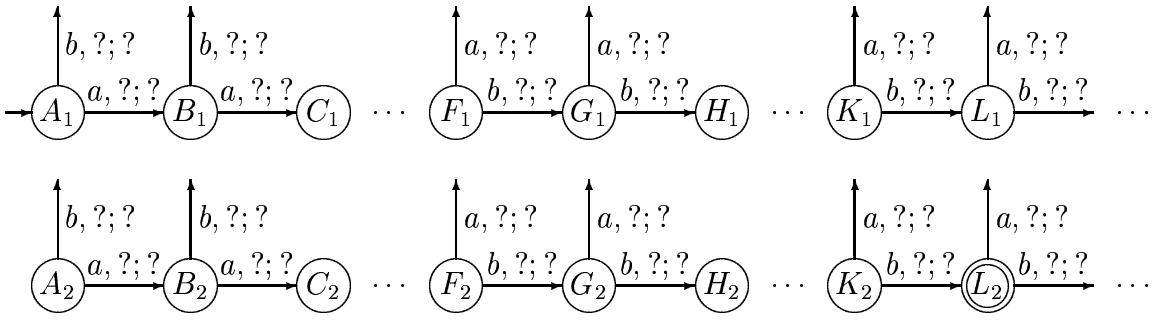


Supposons que pour accepter $a^{40}b^{40}$, M passe de l'état A à l'état G. Il commence par lire 40 a, en manipulant la pile de quelque façon, puis il lit 40 b, en manipulant la pile de manière appropriée. S'il ne rencontre pas le bon symbole, il fait autre chose (puisque M est déterministe, il doit y avoir une transition définie quel que soit le symbole d'entrée). Supposons que pour accepter $a^{40}b^{80}$, il passe de l'état A à l'état L. Ce faisant, il doit nécessairement passer par l'état G, puisqu'il doit d'abord lire $a^{40}b^{40}$ et qu'il est déterministe.

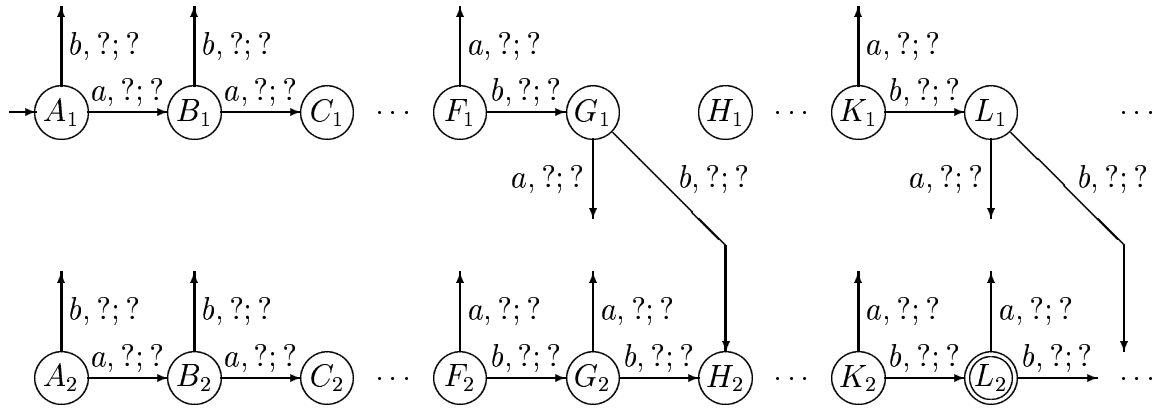
Faisons maintenant deux copies M_1 et M_2 de M.



Nous dirons que les états X_1 et X_2 sont cousins, où $X \in \{A, B, C, \ldots\}$. Enlevons le statut d'état final aux états finaux de M_1 ainsi qu'à G_2, et le statut d'état initial à l'état initial de M_2.



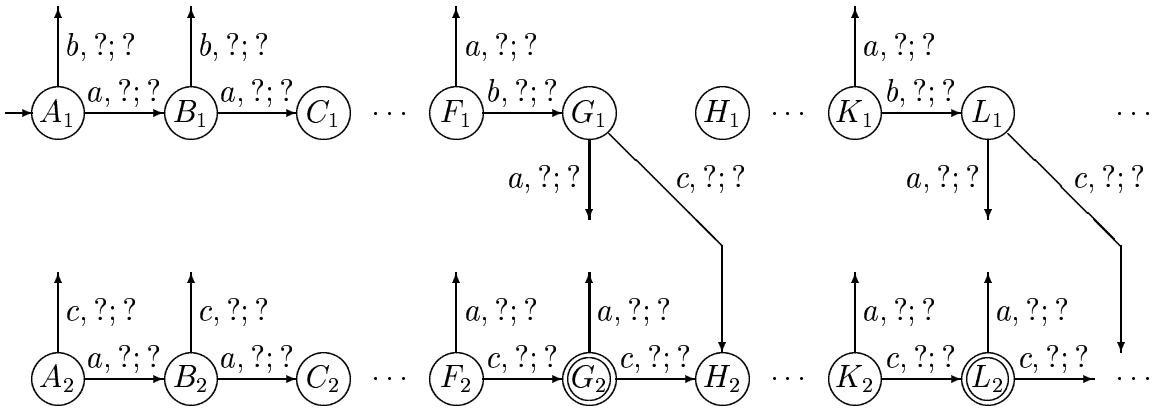
Considérons chaque arc qui quitte un ancien état final de M_1. Si la destination de cet arc est l'état X_1, changeons la destination de l'arc pour l'état X_2 (le cousin de X_1). Les automates M_1 et M_2 forment maintenant un seul automate.



Ce nouvel automate lit la séquence $a^{40}b^{40}$ de la même manière que M_1, c'est-à-dire en faisant les mêmes transitions et en atteignant l'état G_1 ; cependant, il ne l'accepte pas, puisque l'état G_1 n'est plus final. Que se passe-t-il si l'entrée est $a^{40}b^{80}$? Il lit d'abord $a^{40}b^{40}$, se retrouvant ainsi dans l'état G_1. A la lecture du b suivant, l'automate combiné passe dans l'état H_2, à cause de la modification que nous venons de faire. Puis la lecture des 39 autres symboles b l'amène dans l'état L_2, où il accepte $a^{40}b^{80}$.

Comme il n'y a rien de particulier avec le chiffre 40, on voit qu'en fait cet automate accepte $\{a^n b^{2n} : n \in \mathbf{N}^+\}$.

Faisons une dernière modification : Transformons toutes les transitions qui lisent un b sur le ruban d'entrée et qui mènent à un état de M_2 (incluant celles qui quittent M_1) de sorte qu'elles lisent maintenant un c.



C'est maintenant la séquence $a^{40}b^{40}c^{40}$ qui est acceptée. En effet, la séquence $a^{40}b^{40}$ mène de l'état A_1 à l'état G_1 (elle n'est pas acceptée). A la lecture du premier c, l'automate combiné passe dans l'état H_2. Puis la lecture des 39 autres symboles c l'amène dans l'état L_2, où il accepte $a^{40}b^{40}c^{40}$.

En généralisant ce que nous venons de faire à des séquences autres que $a^{40}b^{40}c^{40}$, nous voyons que ce dernier automate accepte

$$\{a^n b^n c^n : n \in \mathbf{N}^+\}.$$

C'est donc l'automate M' annoncé dans la preuve ci-dessus. □

On appelle la classe des langages reconnus par les automates à pile déterministes les langages *non contextuels déterministes*. Parce que cette classe n'englobe pas tous les langages non contextuels, il existe des langages que les analyseurs syntaxiques déterministes simulant des automates à pile ne peuvent reconnaître. On imagine l'impact que cela peut avoir sur la conception des langages de programmation.

On peut aussi démontrer [Brookshear89, page 108] qu'il existe des langages reconnaissables par des automates à pile déterministes qui ne peuvent être reconnus par des automates à pile déterministes qui vident leur pile avant d'accepter ; c'est-à-dire que

$$\begin{aligned} &\{L(M) : M \text{ est un automate à pile déterministe qui vide sa pile avant d'accepter}\} \\ \neq\ &\{L(M) : M \text{ est un automate à pile déterministe}\} \end{aligned}$$

Un exemple d'un tel langage est

$$\{a^n : n \in \mathbf{N}^+\} \cup \{a^n b^n : n \in \mathbf{N}^+\}.$$

Nous avons donc la hiérarchie de langages non contextuels suivante.

{Langages acceptés par des automates à pile déterministes qui vident leur pile avant d'accepter}
$\subset$ {Langages non contextuels déterministes}
$\subset$ {Langages non contextuels}

2.51 Exercice.

1. Donnez un exemple d'un langage non contextuel déterministe.
2. Donnez un exemple d'un langage non contextuel qui n'est pas un langage non contextuel déterministe.

□

La puissance des automates à pile déterministes qui vident leur pile avant d'accepter est relativement faible. Dans la prochaine section, nous verrons comment améliorer leur performance en leur permettant d'examiner, avant de les lire, quelques symboles de la chaîne d'entrée.

2.3.3 Problèmes

1. Construisez un automate à pile déterministe qui reconnaît le langage sur l'alphabet $\{a, b\}$ dont les éléments sont les séquences qui contiennent le même nombre de a que de b.
2. Construisez un automate à pile déterministe M tel que $L(M)$ est le langage généré par la grammaire ci-dessous. Le symbole initial est S.

   $$S \to abN \qquad N \to cS \qquad N \to c \qquad N \to \lambda$$

3. Utilisez le corollaire 2.42 pour montrer que le langage

   $$\{a^p : p \in \mathbf{N} \wedge p \text{ est premier}\}$$

   n'est pas un langage non contextuel.

2.4 Analyse syntaxique $LL(k)$

OBJECTIF SPÉCIFIQUE :

1. Construire et utiliser des tables d'analyse $LL(1)$.

2.4.1 Le processus d'analyse LL

Dans cette section et la suivante, nous allons apprendre comment développer des procédures d'analyse syntaxique à partir d'une grammaire donnée. Dans la présente section, le processus d'analyse étudié s'appelle $LL(k)$.

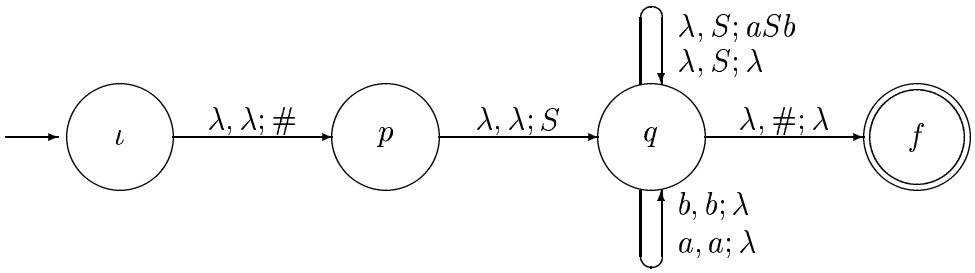
Les caractéristiques du processus d'analyse $LL(k)$ sont les suivantes :

- la lecture de l'entrée se fait de la gauche vers la droite (c'est la raison du premier L (« left ») dans LL) ;
- les dérivations sont des dérivations à gauche (c'est la raison du deuxième L) ;
- l'analyse se fait avec consultation, sans lecture, des k prochains symboles de la séquence d'entrée.

La transformation d'une grammaire en automate peut se faire en utilisant les techniques du théorème 2.26. Par exemple, pour une grammaire constituée des productions

$$S \to aSb \qquad S \to \lambda$$

l'automate est



Dans l'état q,

- si un symbole non terminal se trouve au sommet de la pile, il est remplacé par le côté droit d'une règle ayant ce symbole non terminal comme côté gauche.
- si le sommet de la pile est un symbole terminal égal au symbole d'entrée, ce symbole est lu et dépilé.

L'automate ci-dessus est non déterministe :

1. Il n'est pas totalement défini. C'est facile à changer en ajoutant un état poubelle qui reçoit tous les cas d'erreur (c'est-à-dire les cas non traités par l'automate).

2. Il y a parfois un choix de transitions. Dans l'état q, si le sommet de la pile est S, les transitions $(q, \lambda, S; q, aSb)$ et $(q, \lambda, S; q, \lambda)$ sont toutes deux possibles. Dans ce cas, *la connaissance du prochain symbole d'entrée permet de déterminer la transition à prendre* :
   - si le symbole à lire est a, il faut faire la transition $(q, \lambda, S; q, aSb)$, puisqu'alors un a se retrouve sur le dessus de la pile.
   - si le symbole à lire est un b, il faut choisir l'autre transition. Cela fait disparaître le symbole S du sommet de la pile. Sous ce S il y a peut-être un b.

2.4.2 Tables d'analyse LL

On peut indiquer les transitions à faire au moyen d'une table d'analyse LL. Voyons comment construire ces tables dans le cas de l'analyse $LL(1)$.

Une table d'analyse $LL(1)$ est construite comme suit.

- Les rangées de la table sont étiquetées par les symboles non terminaux de la grammaire.
- Les étiquettes des colonnes sont les symboles terminaux de la grammaire ainsi qu'un symbole spécial FIN. Ce dernier représente un marqueur de fin de chaîne.
- L'entrée (X, y) de la table indique l'action à faire lorsque le symbole non terminal X est au sommet de la pile et que le prochain symbole à lire est un y (incluant FIN). Si le non terminal X doit être remplacé par le côté droit d'une règle de la grammaire, le côté droit de cette règle constitue l'entrée (X, y) ; autrement, l'entrée est le mot *erreur*.

2.52 Exemple. Voici une grammaire et la table d'analyse $LL(1)$ correspondante :

$$\begin{array}{lcl} S & \to & zMNz \\ M & \to & aMa \\ M & \to & z \\ N & \to & bNb \\ N & \to & z \end{array}$$

	a	b	z	FIN
S	*erreur*	*erreur*	$zMNz$	*erreur*
M	aMa	*erreur*	z	*erreur*
N	*erreur*	bNb	z	*erreur*

- L'entrée (S, a) est *erreur*, car le symbole S peut seulement être remplacé par le côté droit $zMNz$, ce qui amène un z au sommet de la pile. Ce z peut être dépilé seulement si le prochain symbole à lire est z.
- L'entrée (M, a) est aMa, car c'est la seule manière d'amener un a au sommet de la pile.
- Puisque la grammaire n'a pas de règle-λ (i.e. $S \to \lambda$, $M \to \lambda$ ou $N \to \lambda$), on a toujours *erreur* dans chacune des entrées de la colonne étiquetée FIN. En effet, s'il y a un symbole non terminal au sommet de la pile alors que la séquence d'entrée est toute lue, cela constitue une erreur, puisque le remplacement du

symbole non terminal par le côté droit d'une règle introduit un symbole terminal sur la pile. Ce symbole terminal ne pourra être dépilé, puisque la séquence d'entrée est toute lue ; en conséquence, le symbole #, qui se trouve en dessous, ne pourra être dépilé et l'automate ne pourra atteindre l'état final (voir par exemple l'automate de la page 156).

- Etc.

□

Voici une procédure d'analyse $LL(1)$ qui simule le comportement d'un automate comme celui de la page 156 en utilisant la table $LL(1)$ correspondante pour déterminer les actions à prendre. Elle contient en plus des instructions d'impression ; cela sera utile pour donner des exemples d'exécution.

```
PROCÉDURE LL(1) ;

empiler le symbole initial de la grammaire ;
lire (Symbole) ;
tant que la pile n'est pas vide faire
   imprimer l'état de la pile, la valeur de Symbole et ce qui reste à lire ;
   cas sommet de la pile
      terminal : si sommet = Symbole
         alors dépiler et lire (Symbole)
         sinon rejeter la séquence d'entrée et sortir ;
      non terminal : si table[sommet,Symbole] ≠ erreur
         alors dépiler puis empiler table[sommet,Symbole]
         sinon rejeter la séquence d'entrée et sortir
   fin cas
fin tant que
imprimer l'état de la pile, la valeur de Symbole et ce qui reste à lire ;
si Symbole ≠ FIN alors rejeter la séquence d'entrée et sortir
```

Remarquons que l'action de lecture d'un automate est différente de l'action de lecture de la procédure ci-dessus. Lorsqu'un automate lit un symbole, il le traite par la même occasion et s'en débarrasse (il ne le revoit plus). Par contre, lorsque la procédure ci-dessus lit un symbole, elle le place dans la variable Symbole, qu'elle peut consulter tant qu'une autre valeur n'est pas affectée à Symbole. La lecture, pour l'automate, est une consommation du symbole. Pour la procédure, c'est une façon de regarder le symbole ; du point de vue de l'automate, l'action de la procédure revient à regarder le prochain symbole à lire, sans le lire (d'où le « 1 » dans « $LL(1)$ »).

2.53 Exemple. Voici comment la séquence *zazazz* est reconnue par la procédure

LL(1) lorsqu'elle utilise la table LL de l'exemple 2.52.

	Pile	Symbole	À lire
1	S	z	$azazz$FIN
2	$zMNz$	z	$azazz$FIN
3	MNz	a	$zazz$FIN
4	$aMaNz$	a	$zazz$FIN
5	$MaNz$	z	azzFIN
6	$zaNz$	z	azzFIN
7	aNz	a	zzFIN
8	Nz	z	zFIN
9	zz	z	zFIN
10	z	z	FIN
11		FIN	

Remarquez comment l'acceptation de $zazazz$ correspond à une dérivation à gauche de cette séquence :

$$
\begin{aligned}
& S \\
\Rightarrow \quad & \langle \text{ Règle } S \to zMNz, \text{ appliquée au } S \text{ de l'étape 1. } \rangle \\
& zMNz \\
\Rightarrow \quad & \langle \text{ Règle } M \to aMa, \text{ appliquée au } M \text{ de l'étape 3. } \rangle \\
& zaMaNz \\
\Rightarrow \quad & \langle \text{ Règle } M \to z, \text{ appliquée au } M \text{ de l'étape 5. } \rangle \\
& zazaNz \\
\Rightarrow \quad & \langle \text{ Règle } N \to z, \text{ appliquée au } N \text{ de l'étape 8. } \rangle \\
& zazazz
\end{aligned}
$$

□

2.54 Exercice. Voici une grammaire $G = (\{S\}, \{a, b, c\}, S, R)$ où R est l'ensemble des règles qui suivent.

$S \to aSc$ $\quad$ $S \to bSc$ $\quad$ $S \to \lambda$

1. Dites quel est $L(G)$.
2. Construisez la table $LL(1)$ de cette grammaire.
3. Donnez les résultats imprimés par la procédure LL(1) de la page 158 lorsqu'elle analyse les séquences suivantes au moyen de la table construite en 2. Dites si les séquences sont acceptées ou rejetées.
   (a) abcc
   (b) abc

□

2.55 Définition. Une grammaire est dite $LL(1)$ si le langage qu'elle génère peut être reconnu par un analyseur $LL(1)$. □

Certaines grammaires non contextuelles nécessitent plus d'un symbole de prévision pour qu'un analyseur LL puisse reconnaître leur langage. Voici un exemple d'une telle grammaire.

$S \to aSc$ $\quad$ $S \to abTbc$ $\quad$ $T \to \lambda$

Lorsqu'un S est au sommet de la pile, on voit que même si on sait que le prochain symbole d'entrée est a, on ne peut choisir la règle à appliquer. Il faut connaître les deux prochains symboles à l'entrée. La grammaire est dite $LL(2)$. Voici la table $LL(2)$ correspondante (toutes les entrées de la ligne T, sauf celle de la colonne bc, pourraient être remplacées par *erreur*).

	aa	ab	ac	ba	bb	bc	ca	cb	cc	FIN
S	aSc	$abTbc$	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*
T	λ	λ	λ	λ	λ	λ	λ	λ	λ	λ

La classe des langages $LL(k)$ est un sous-ensemble strict de la classe des langages non contextuels *déterministes*. En effet il existe des langages non contextuels déterministes qui ne sont $LL(k)$ pour aucun k. Le langage généré par la grammaire G suivante en est un exemple :

$$S \to T \qquad T \to aT \qquad V \to aVb$$
$$S \to V \qquad T \to \lambda \qquad V \to \lambda$$

$$L(G) = \{a^n : n \in N\} \cup \{a^n b^n : n \in N\}.$$

Supposons que $L(G)$ puisse être analysé par un analyseur LL utilisant 100 symboles de prévision. Si on soumet à cet analyseur une chaîne qui débute par 100 a consécutifs, celui-ci est incapable de choisir dès le départ quelle production appliquer ($S \to T$ ou $S \to V$) à l'axiome S sur sa pile. Dans l'espoir de pouvoir faire un choix, il consulte le premier symbole de prévision, puis le deuxième, ensuite le troisième, etc., jusqu'au centième, toujours sans savoir laquelle des deux productions choisir.

2.4.3 Problèmes

1. Construisez une table d'analyse $LL(1)$ pour la grammaire

$$S \to aSc \qquad S \to b$$

2. Soit la grammaire non contextuelle $G = (\{E, F, T\}, \{a, b, +, -, (,)\}, E, R)$ où R est l'ensemble des règles suivantes.

$$\begin{array}{lll} E \to TF & F \to +TF & T \to (E) \\ & F \to -TF & T \to a \\ & F \to \lambda & T \to b \end{array}$$

Cette grammaire génère des expressions arithmétiques (d'où le E comme symbole de départ). Le F génère les « fins d'expressions » et le T génère les termes.

(a) Construisez une table d'analyse $LL(1)$ pour la grammaire G.

(b) Soient les séquences $u = ((a + b) - (a - b))$ et $v = (a + b)(a - b)$. Utilisez u et v comme données d'entrée pour la procédure LL(1) de la page 158. Donnez les informations demandées dans le programme. La table $LL(1)$ utilisée par le programme est évidemment celle que vous avez trouvée en (a).

(c) Dites si u et v sont acceptées ou non par le programme (c'est-à-dire si $u, v \in L(G)$). Pour chaque séquence acceptée, donnez la dérivation impliquée par l'exécution du programme.

2.5 Analyse syntaxique $LR(k)$

OBJECTIF SPÉCIFIQUE

1. Utiliser des tables d'analyse $LR(1)$.

2.5.1 Le processus d'analyse LR

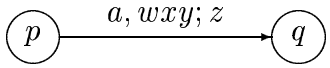
À la section précédente, nous avons étudié le mécanisme d'analyse syntaxique $LL(k)$. Nous introduisons maintenant un autre type d'analyse syntaxique, appelé $LR(k)$. Ce processus d'analyse se caractérise par :

- la lecture de l'entrée se fait de la gauche vers la droite (c'est la raison du L (« left ») dans « $LR(k)$ » ;
- les dérivations sont des dérivations à droite (c'est la raison du R (« right »)) ;
- l'analyse se fait avec consultation, sans lecture, des k prochains symboles de la séquence d'entrée.

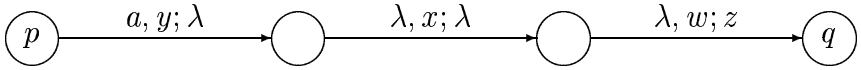
Nous savons déjà comment construire un automate reconnaissant le langage généré par une grammaire non contextuelle G (théorème 2.26). Cette construction a été utile pour l'étude du processus LL. Pour l'approche LR, il faut utiliser une construction différente. Avant de décrire cette nouvelle méthode, donnons-nous une convention qui

permettra d'utiliser des transitions qui dépilent plusieurs symboles (nous en avons déjà une qui permet des transitions empilant plusieurs symboles).

2.56 Notation. Nous utiliserons des transitions de la forme $(p, a, wxy; q, z)$, c'est-à-dire que plusieurs symboles peuvent être dépilés à la fois. La section d'automate correspondant à une telle transition est la suivante.



Un tel diagramme est en fait une abréviation du diagramme qui suit.



De ce deuxième diagramme, on voit que le dépilement se fait dans l'ordre y, x, w, c'est-à-dire que la séquence wxy est dépilée de la droite vers la gauche, tout comme elle est empilée de la droite vers la gauche par la transition $(p, a, z; q, wxy)$. □

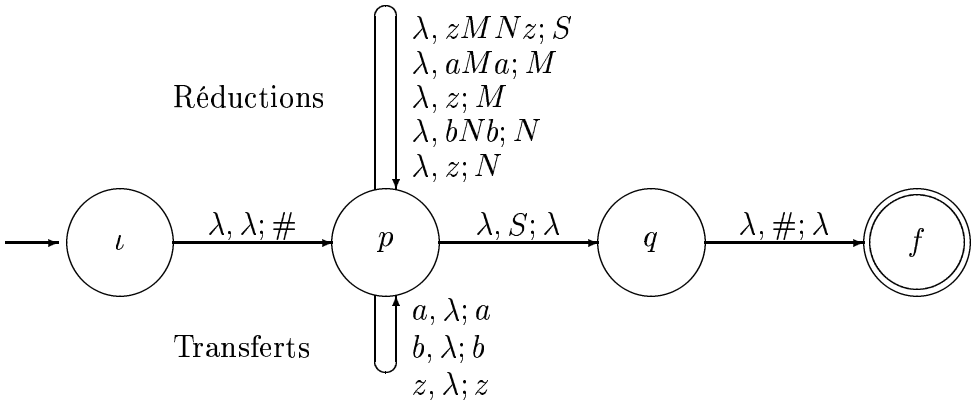
Voici maintenant, en cinq étapes, une autre manière de construire un tel automate. Nous allons l'illustrer avec la grammaire qui suit :

$$\begin{array}{lll} S \to zMNz & M \to aMa & N \to bNb \\ & M \to z & N \to z \end{array}$$

1. Notre automate comporte quatre états : ι, p, q, f. L'état initial est ι et le seul état final est f.
2. Ajouter les transitions $(\iota, \lambda, \lambda; p, \#)$ et $(q, \lambda, \#; f, \lambda)$. Comme d'habitude, celles-ci servent à marquer le fond de la pile.
3. On introduit, pour chaque terminal x de l'alphabet, la transition $(p, x, \lambda; p, x)$. Ces transitions sont appelées des *transferts* puisque chacune d'elles équivaut au transfert sur la pile du symbole à l'entrée.
4. Pour chaque règle de la forme $N \to w$, on ajoute la transition $(p, \lambda, w; p, N)$. Ces transitions sont appelées des *réductions*, car elles permettent de réduire une séquence de symboles (w) à un seul symbole (N).

   Notez que $|w| \geq 0$. Lorsque $|w| = 0$, c'est-à-dire pour $N \to \lambda$, nous avons la transition $(p, \lambda, \lambda; p, N)$.
5. On ajoute la transition $(p, \lambda, S; q, \lambda)$, où S est l'axiome de la grammaire.

Le diagramme de transitions de l'automate ainsi construit est :



Voici la suite de configurations menant à l'acceptation de la chaîne *zazabzbz*. Afin que les côtés droits des règles présents dans la pile se lisent facilement (de gauche à droite), nous revenons à la convention utilisée à la section 2.1 et nous empilons les symboles de gauche à droite.

Étape	État	Pile	À lire
0	ι		$zazabzbz$
1	p	$\#$	$zazabzbz$
2	p	$\#z$	$azabzbz$
3	p	$\#za$	$zabzbz$
4	p	$\#zaz$	$abzbz$
5	p	$\#zaM$	$abzbz$
6	p	$\#zaMa$	$bzbz$
7	p	$\#zM$	$bzbz$
8	p	$\#zMb$	zbz
9	p	$\#zMbz$	bz
10	p	$\#zMbN$	bz
11	p	$\#zMbNb$	z
12	p	$\#zMN$	z
13	p	$\#zMNz$	
14	p	$\#S$	
15	q	$\#$	
16	f		

La dérivation effectuée par l'automate est une dérivation à droite, comme on peut le constater en lisant la table ci-dessus de bas en haut :

$$
\begin{aligned}
& S \\
\Rightarrow \quad & \qquad \langle\ S \to zMNz : \text{réduction de } zMNz \text{ par } S, \text{ lignes 13 et 14.}\ \rangle
\end{aligned}
$$

$zMNz$

$\Rightarrow$ $\langle$ $N \to bNb$: réduction de bNb par N, lignes 11 et 12. $\rangle$

$zMbNbz$

$\Rightarrow$ $\langle$ $N \to z$: réduction de z par N, lignes 9 et 10. $\rangle$

$zMbzbz$

$\Rightarrow$ $\langle$ $M \to aMa$: réduction de aMa par M, lignes 6 et 7. $\rangle$

$zaMabzbz$

$\Rightarrow$ $\langle$ $M \to z$: réduction de z par M, lignes 4 et 5. $\rangle$

$zazabzbz$

Remarquons que l'automate est non déterministe : souvent, il doit choisir la bonne transition parmi celles qui sont possibles. Par exemple, pour passer de la ligne 2 à la ligne 3, l'automate a choisi le transfert $(p, a, \lambda; p, a)$, mais il aurait aussi pu faire la réduction $(p, \lambda, z; p, M)$ ou la réduction $(p, \lambda, z; p, N)$; cependant, comme on le vérifie facilement, ces choix mènent à un impasse. De même, pour passer de la ligne 4 à la ligne 5, l'automate aurait pu choisir un transfert ou la réduction $(p, \lambda, z; p, N)$ plutôt que la réduction $(p, \lambda, z; p, M)$; le résultat, ici aussi, aurait été une impasse.

2.57 Exercice. Soit la grammaire G :

$S \to aSc$ $\quad$ $S \to bSc$ $\quad$ $S \to \lambda$

1. Construisez un automate à pile acceptant $L(G)$ en utilisant la méthode présentée à la page 163.
2. Donnez la suite des configurations qui mènent à l'acceptation de *babccc*.
3. Quelle est la dérivation qu'effectue l'automate lorsqu'il analyse la séquence *babccc* ?

□

2.5.2 Tables d'analyse LR

Nous voulons construire une procédure d'analyse syntaxique basée sur l'automate précédent. Celui-ci présente deux problèmes.

1. Il est non déterministe. Pour remédier à cette lacune, permettons-lui de consulter (mais sans les lire) les k prochains symboles d'entrée.
2. La détermination de la séquence au sommet de la pile nécessite une fouille de celle-ci. Par exemple, pour savoir si la réduction $(p, \lambda, aMa; p, M)$ est possible, l'automate doit examiner les trois symboles au sommet de la pile. Pour éviter cette fouille, on empile en plus des symboles de G des nombres appelés *jetons*. Un jeton caractérise une séquence de symboles de G ; par exemple, la présence d'un 5 au sommet de la pile pourrait indiquer que la séquence aMa se trouve en-dessous.

Pour les langages déterministes non contextuels, l'information nécessaire peut être mise dans une table d'analyse. Une routine générique peut ensuite utiliser cette table pour analyser une séquence du langage.

2.58 Exemple. Voici la table d'analyse $LR(1)$ correspondant à la grammaire

$S \to zMNz$ $\quad$ $M \to aMa$ $\quad$ $N \to bNb$
$\quad\quad\quad\quad\quad\quad$ $M \to z$ $\quad\quad$ $N \to z$

(voyez [Brookshear89, page 123]).

	a	b	z	FIN	S	M	N
1			t 2		14		
2	t 3		t 7			4	
3	t 3		t 7			8	
4		t 5	t 9				6
5		t 5	t 9				10
6			t 11				
7	$M \to z$	$M \to z$	$M \to z$				
8	t 12						
9		$N \to z$	$N \to z$				
10		t 13					
11				$S \to zMNz$			
12	$M \to aMa$	$M \to aMa$	$M \to aMa$				
13		$N \to bNb$	$N \to bNb$				
14				accepte			

Les colonnes de la table sont étiquetées avec les symboles de la grammaire (terminaux et non terminaux) et le marqueur de fin de chaîne FIN. Il y a quatre types d'entrées dans cette table.

1. Les cases vides : elles indiquent une erreur (rejet de la séquence).
2. Les cases contenant seulement un jeton (chiffre).
3. Les case contenant une règle de la grammaire (réductions).
4. Les case avec un « t » suivi d'un jeton (transferts).

□

Expliquons maintenant comment une table comme celle de l'exemple 2.58 est utilisée (voir la procédure LR(1) qui suit). L'analyse d'une séquence commence en empilant le jeton 1. Par dessus chaque symbole de la grammaire qui sera empilé, nous empilerons un jeton ; ainsi la pile contiendra, en alternance, des symboles de la grammaire et des jetons. Chaque jeton représentera une partie de la structure interne de la pile, ce qui évitera d'y rechercher des patrons particuliers. La variable Jeton contient la valeur courante du jeton et la variable Symbole contient la valeur du symbole à l'entrée (examen sans consommation, comme dans le cas de l'analyse LL).

A chaque itération de l'algorithme, la case (Jeton,Symbole) de la table est examinée.

1. Si Table [Jeton,Symbole] est vide, la séquence doit être rejetée.
2. Si Table [Jeton,Symbole] est « accepte », la séquence est acceptée, à la condition qu'elle ait été toute lue.
3. Si Table [Jeton,Symbole] est une règle de grammaire de la forme $X \to w$, une réduction doit être effectuée. Les symboles formant la séquence w doivent être dépilés, ainsi que les jetons qui les recouvrent. Ensuite, le symbole X est empilé. Puis, par dessus ce symbole, un jeton est empilé ; sa valeur est trouvée dans la table à la colonne X et à la ligne correspondant à la valeur du jeton sous X.
4. Si Table [Jeton,Symbole] a la forme « t n », un transfert doit être fait. Le symbole courant (contenu dans Symbole) est empilé et le jeton n est empilé par dessus. Finalement, un nouveau symbole est lu (dans Symbole).

Voici la procédure LR(1) qui met en oeuvre l'analyse syntaxique $LR(1)$. Dans cette procédure, la notation « EntréeTable.jeton » désigne le jeton n lorsque l'entrée a la forme « t n ».

```
PROCÉDURE LR(1)

Jeton := 1 ;
empiler (Jeton) ;
lire (Symbole) ;
EntréeTable := Table [Jeton,Symbole] ;
tant que EntréeTable ≠ "accepte" faire
    donner l'état de la pile, ce qui reste à lire
```

```
        ainsi que les valeurs de Symbole, Jeton et EntréeTable ;
    si EntréeTable est un transfert alors
        début
        empiler (Symbole) ; Jeton := EntréeTable.jeton ;
        empiler (Jeton) ; lire (Symbole)
        fin
    sinon si EntréeTable est une règle alors   (* réduction *)
        début
        dépiler (côté droit de EntréeTable) ;   (* ainsi que les jetons *)
        Jeton := sommet (pile) ;   (* ne pas dépiler *)
        empiler (côté gauche de EntréeTable) ;
        Jeton := Table [Jeton, côté gauche de EntréeTable] ;
        empiler (Jeton)
        fin
    sinon si EntréeTable est vide alors rejeter et sortir ;
    EntréeTable := Table [Jeton,Symbole] ;
fin tant que ;
Donner l'état de la pile, ce qui reste à lire
    ainsi que les valeurs de Symbole, Jeton et EntréeTable ;
si Symbole ≠ FIN alors rejeter et sortir ;
vider la pile
```

Pour mieux comprendre les détails de cet algorithme, exécutons-le sur un exemple.

2.59 Exemple. Montrons comment la séquence *zazabzbz* est acceptée par la procédure LR(1) ci-dessus lorsqu'elle utilise la table d'analyse $LR(1)$ de l'exemple 2.58.

	Pile	À lire	Symbole	Jeton	EntréeTable
1	1	*azabzbz*	*z*	1	t 2
2	1*z*2	*zabzbz*	*a*	2	t 3
3	1*z*2*a*3	*abzbz*	*z*	3	t 7
4	1*z*2*a*3*z*7	*bzbz*	*a*	7	$M \to z$
5	1*z*2*a*3*M*8	*bzbz*	*a*	8	t 12
6	1*z*2*a*3*M*8*a*12	*zbz*	*b*	12	$M \to aMa$
7	1*z*2*M*4	*zbz*	*b*	4	t 5
8	1*z*2*M*4*b*5	*bz*	*z*	5	t 9
9	1*z*2*M*4*b*5*z*9	*z*	*b*	9	$N \to z$
10	1*z*2*M*4*b*5*N*10	*z*	*b*	10	t 13
11	1*z*2*M*4*b*5*N*10*b*13		*z*	13	$M \to bNb$
12	1*z*2*M*4*N*6		*z*	6	t 11
13	1*z*2*M*4*N*6*z*11		FIN	11	$S \to zMNz$
14	1*S*14		FIN	14	accepte

La séquence est acceptée, puisqu'elle est toute lue et que l'entrée « accepte » a été atteinte.

À la page 164, on donne la suite de configurations menant à l'acceptation de *zazabzbz* par l'automate de la page 164. On peut constater la similitude entre cette suite de configurations et les résultats de l'exécution de la procédure LR(1) présentés ci-dessus. □

Nous n'étudierons pas la manière de construire les tables d'analyse $LR(k)$. Il s'agit d'un processus relativement complexe, qui fait l'objet des cours sur la compilation des langages informatiques.

On montre que la classe des langages $LR(k)$ (en prenant tous les $k \in \mathbf{N}$) est la même que la classe des langages non contextuels déterministes. Donc les analyseurs $LR(k)$ sont plus puissants que les analyseurs $LL(k)$.

2.60 Exercice. Utilisez la procédure LR(1) avec la table de l'exemple 2.58 pour déterminer si la séquence *zzbzbzz* est acceptée. Donnez les résultats demandés par la procédure.

□

2.5.3 Problème

1. Soit la grammaire non contextuelle $G = (\{E, T\}, \{a, b, +, -, (,)\}, E, R)$ où R est l'ensemble des règles suivantes.

   $E \to E + T$ $\quad$ $T \to (E)$
   $E \to E - T$ $\quad$ $T \to a$
   $E \to T$ $\quad$ $T \to b$

   Cette grammaire génère des expressions arithmétiques (d'où le E comme symbole de départ). (Voyez aussi le problème 2 de la section 2.4, page 162). Ce qui suit est la table d'analyse $LR(1)$ correspondant à G.

	a	b	$+$	$-$	(	)	FIN	E	T
1	t 6	t 7			t 10			2	5
2			t 3	t 4			accepte		
3	t 6	t 7			t 10				8
4	t 6	t 7			t 10				9
5			$E \to T$	$E \to T$		$E \to T$	$E \to T$		
6			$T \to a$	$T \to a$		$T \to a$	$T \to a$		
7			$T \to b$	$T \to b$		$T \to b$	$T \to b$		
8			$E \to E + T$	$E \to E + T$		$E \to E + T$	$E \to E + T$		
9			$E \to E - T$	$E \to E - T$		$E \to E - T$	$E \to E - T$		
10	t 6	t 7			t 10			11	5
11			t 3	t 4		t 12			
12			$T \to (E)$	$T \to (E)$		$T \to (E)$	$T \to (E)$		

   (a) Soient les séquences $u = ((a + b) - (a - b))$ et $v = (a + b)(a - b)$. Utilisez u et v comme données d'entrée pour la procédure LR(1) de la page 167. La table d'analyse $LR(1)$ utilisée par la procédure est la table ci-dessus. Donnez les informations demandées par le programme.

   (b) Dites si u et v sont acceptées ou non par la procédure (c'est-à-dire si $u, v \in L(G)$). Pour chaque séquence acceptée, donnez la dérivation impliquée par l'exécution du programme.

Chapitre 3

Machines de Turing

OBJECTIF GÉNÉRAL

1. Comprendre les capacités et les limites des machines de Turing (et donc des ordinateurs) en tant que dispositifs de reconnaissance de langages. Faire le lien avec les grammaires à structure de phrase.

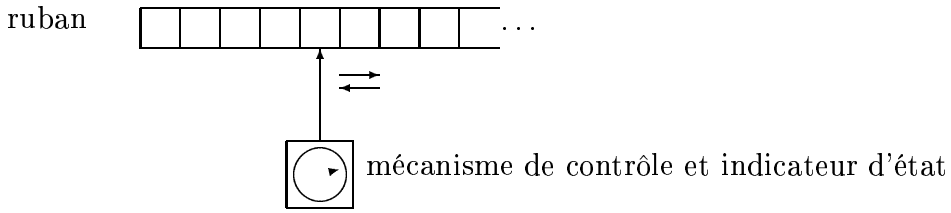
3.1 Machines de Turing

OBJECTIF SPÉCIFIQUE

1. Démontrer le fonctionnement d'une machine de Turing donnée en énumérant les configurations consécutives à une configuration initiale donnée.

Dans ce chapitre, nous étudierons une nouvelle classe de machines, les machines de Turing, ainsi que les langages qu'elles acceptent, les langages à structures de phrase. Les machines de Turing sont les machines les plus puissantes. Elles ont été inventées par Alan M. Turing en 1936. Elles sont semblables aux automates finis, sauf qu'elles peuvent lire et écrire sur leur ruban, et que leur tête de lecture/écriture peut être déplacée dans les deux directions.

Voici une représentation schématique d'une machine de Turing :



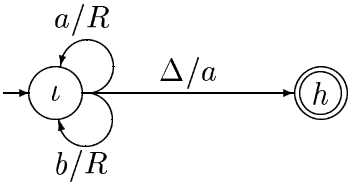
Comme pour les automates finis, le ruban est infini vers la droite. Les différences avec les automates finis sont :

1. La tête peut se déplacer dans les deux sens.
2. C'est une tête de lecture/écriture.
3. Il y a un seul état final, qui est un état d'arrêt (aussitôt que la machine l'atteint, elle s'arrête).

A chaque étape, la machine

1. lit un symbole ;
2. fait une transition d'état (elle peut rester dans le même état) ;
3. fait l'une des trois actions suivantes :
   - écriture d'un symbole ;
   - déplacement de la tête vers la droite ;
   - déplacement de la tête vers la gauche.

3.1 Exemple. Voici un diagramme de transitions d'une machine de Turing.



- ι est l'état initial.
- h est l'état final (état d'arrêt).
- Δ/a signifie *lire* Δ, *puis écrire* a.
- a/R signifie *lire* a, *puis déplacer la tête vers la droite.*
- b/R signifie *lire* b, *puis déplacer la tête vers la droite.*

□

Voici maintenant la définition formelle d'une machine de Turing.

3.2 Définition. Une machine de Turing est un sixtuplet $(S, \Sigma, \Gamma, \delta, \iota, h)$ où

1. S est un ensemble fini d'états.
2. Σ est l'*alphabet de la machine* ou *alphabet d'entrée*(les séquences d'entrée sont formées avec cet alphabet). L'*espace libre* (ou *blanc*), symbolisé par Δ, ne fait pas partie de Σ, c'est-à-dire qu'on a $\Delta \notin \Sigma$.
3. Γ est l'*alphabet du ruban.*On a $\Sigma \subseteq \Gamma$ et $\Delta \in \Gamma$.
4. δ est la *fonction de transition*

   $$\delta : (S - \{h\}) \times \Gamma \to S \times (\Gamma \cup \{L, R\}) \qquad \text{où } L, R \notin \Gamma.$$

   La signification de δ est la suivante :
   - $\delta(p, x) = (q, y)$ (où $y \in \Gamma$) : si dans l'état p le symbole sous la tête est x, remplacer x par y et passer dans l'état q.
   - $\delta(p, x) = (q, L)$: si dans l'état p le symbole sous la tête est x, déplacer la tête d'une case vers la gauche et passer dans l'état q.
   - $\delta(p, x) = (q, R)$: si dans l'état p le symbole sous la tête est x, déplacer la tête d'une case vers la droite et passer dans l'état q.

   Puisque δ est une fonction, la définition ci-dessus est celle d'une machine déterministe, c'est-à-dire
   - non ambiguë
   - totalement définie (sauf pour l'état h, d'où n'origine aucune transition).
5. ι est l'état initial. Dans les diagrammes de transitions, il sera indiqué de la même manière que pour les automates finis.
6. h est l'état final (état d'arrêt). Dans les diagrammes de transitions, il sera indiqué par un double cercle, comme d'habitude.

□

3.3 Remarque. Une machine de Turing peut

1. se rendre à l'état final et arrêter ;
2. boucler indéfiniment ;
3. terminer anormalement en déplaçant la tête à gauche du ruban.

□

3.4 Remarque. On suppose que Δ représente une case inoccupée du ruban : Initialement, toutes les cases autres que celles qui contiennent la séquence d'entrée contiennent un Δ. Notez aussi qu'il y a un seul état final. □

3.5 Exemple. Avec la machine de l'exemple 3.1, on a

$S = \{\iota, h\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, \Delta\}$
$\delta(\iota, a) = (\iota, R)$
$\delta(\iota, b) = (\iota, R)$
$\delta(\iota, \Delta) = (h, a)$

Tant que cette machine rencontre des a et des b, elle déplace la tête vers la droite. Lorsqu'elle rencontre un blanc, elle passe dans l'état final, en remplaçant ce blanc par un a. □

3.6 Définition. Une *configuration* d'une machine de Turing, c'est la donnée d'un état et une description du contenu du ruban avec une indication de la position de la tête de lecture. La description du ruban se fait en donnant la séquence de symboles du ruban en commençant par la première case. Le symbole sous la tête de lecture/écriture est souligné. Des $\cdots$ à droite de cette séquence indiquent que le reste du ruban contient des blancs. □

3.7 Exemple. Une configuration possible de la machine de l'exemple 3.1 est

$\iota \quad ab\underline{b}\Delta a\Delta\Delta\cdots$

L'état est ι. Le premier symbole du ruban est le a de gauche ; il est suivi d'un b, puis d'un autre b sur lequel se trouve la tête de lecture de la machine.

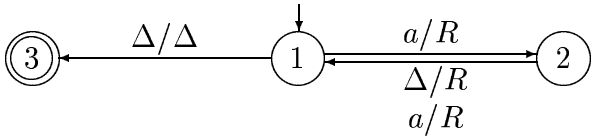
Supposons que la configuration initiale soit $\iota \quad \underline{a}bb\Delta a\Delta\Delta\cdots$ Les configurations successives sont alors

état	ruban	
ι	$\underline{a}bb\Delta a\Delta\Delta\cdots$	
ι	$a\underline{b}b\Delta a\Delta\Delta\cdots$	
ι	$ab\underline{b}\Delta a\Delta\Delta\cdots$	
ι	$abb\underline{\Delta}a\Delta\Delta\cdots$	
h	$abb\underline{a}a\Delta\Delta\cdots$	arrêt

□

La *thèse de Turing* est que tout ce qui est calculable peut être calculé par une machine de Turing. Il n'est pas possible de prouver cette affirmation. Cela dépend de la définition de *calculable*. La thèse de Turing est vraie si *calculer* signifie *manipuler un nombre fini de symboles et produire une réponse après un nombre fini d'étapes.*

3.8 Exercice. Supposons que la configuration initiale de la machine de Turing suivante soit 1 $\underline{a}aa\Delta\Delta\cdots$ Donnez les configurations successives à cette configuration initiale.



□

3.9 Exercice. Construisez une machine de Turing $M = (S, \{a, b\}, \{a, b, \Delta\}, \delta, \iota, h)$ qui arrête si et seulement si le patron *abab* se trouve sur le ruban à droite de la position initiale de la tête de lecture. Donnez la suite de configurations si la configuration initiale est $\iota\ \underline{a}\Delta abab\Delta\cdots$. Que se passe-t-il si la configuration initiale est $\iota\ \underline{a}b\Delta\cdots$?

□

3.2 Construction modulaire

OBJECTIF SPÉCIFIQUE

1. Démontrer le fonctionnement d'une machine de Turing donnée sous la forme d'un schéma de combinaison de machines de Turing en énumérant les configurations du ruban consécutives à une configuration initiale donnée.

Nous allons définir quelques machines de Turing simples qui vont servir de blocs de base pour les constructions ultérieures. Mais d'abord, voyons comment combiner des machines de Turing.

3.2.1 Combinaisons de machines de Turing

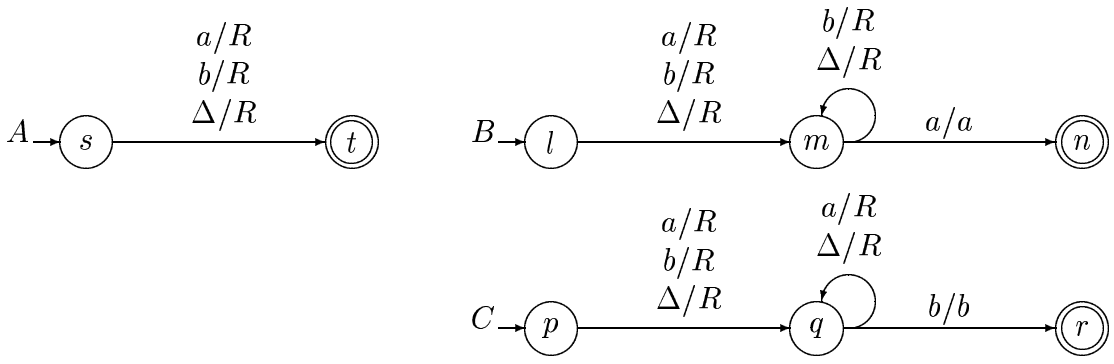
Méthode :

1. Enlever le statut d'état initial à tous les états initiaux, sauf à celui de la machine qui est activée en premier.
2. Enlever le statut d'état final à tous les états finaux et introduire un nouvel état final.
3. Pour chaque ancien état final p et chaque $x \in \Gamma$:
   (a) Si, après avoir atteint l'état p avec symbole courant x (symbole sous la tête de lecture/écriture), la machine de Turing composée doit s'arrêter, ajouter la transition x/x de p vers le nouvel état final.
   (b) Si, après avoir atteint l'état p avec symbole courant x, la machine combinée doit transférer le contrôle à la machine $M = (S, \Sigma, \Gamma, \delta, \iota, h)$, alors ajouter la transition x/z de p vers l'état q de M tel que $\delta(\iota, x) = (q, z)$ (c'est-à-dire qu'on ajoute la transition $\delta(p, x) = (q, z)$). L'ajout de cette transition revient à simuler, dans l'état p, le comportement de M dans l'état ι lors de la lecture d'un x.

3.10 Exemple. Nous avons trois machines de Turing A, B, C avec $\Gamma = \{a, b, \Delta\}$.

- La machine A déplace la tête d'une case vers la droite.
- La machine B trouve le premier a à droite de la cellule courante.
- La machine C trouve le premier b à droite de la cellule courante.

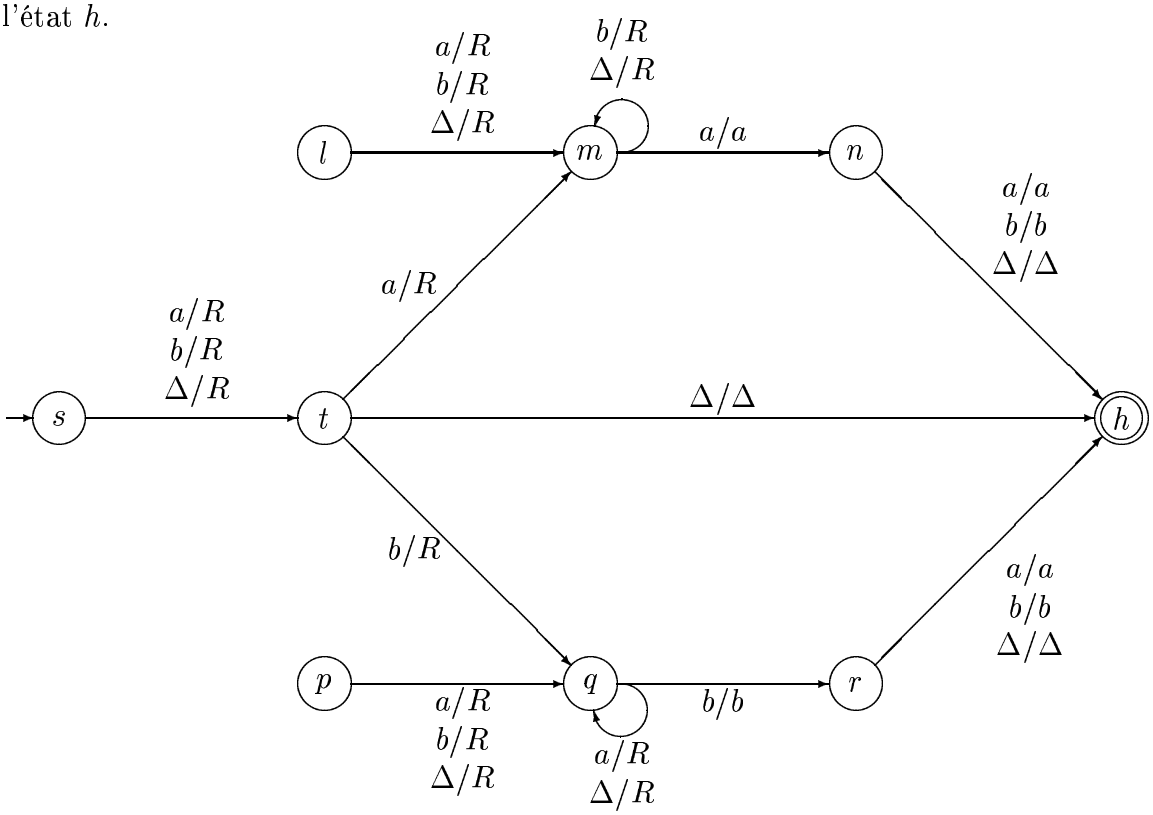
Les diagrammes de transitions de ces machines sont les suivants.



On demande de construire une machine qui trouve la deuxième occurrence du symbole non blanc (i.e. différent de Δ) qui est à droite de la position initiale de la tête de lecture/écriture. Si le symbole à droite de la position initiale de la tête est Δ, la machine combinée doit terminer immédiatement.

La solution est la suivante. Les transitions de l'état n vers l'état h, de l'état r vers l'état h et la transition Δ/Δ de l'état t vers l'état h sont ajoutées en vertu de la partie 3(a) de la méthode ci-dessus, alors que les autres transitions qui partent de l'état t

relèvent de la clause 3(b). Certaines des transitions qui sont ajoutées ne seront jamais utilisées, comme la transition b/b à partir de l'état n (car dans l'état n, le symbole courant est toujours un a). Néanmoins, ces transitions doivent être ajoutées afin que la machine composée soit totalement définie. Dans le cas de la transition b/b, on peut considérer que si l'état n est atteint avec un b comme symbole courant, alors la machine composée doit s'arrêter ; c'est pour cela que cette transition se dirige vers l'état h.



Supposons que la configuration initiale soit

$$s \quad a\Delta b\underline{a}babba\Delta\cdots$$

Le symbole à droite de la tête de lecture/écriture est b. L'occurrence de b cherchée est la suivante : c'est le b qui occupe la septième case du ruban. Voici une suite de configurations qui démontre comment cette occurrence est trouvée.

état	ruban
s	$a\Delta b\underline{a}babba\Delta\cdots$
t	$a\Delta ba\underline{b}abba\Delta\cdots$
q	$a\Delta bab\underline{a}bba\Delta\cdots$
q	$a\Delta babab\underline{b}ba\Delta\cdots$
r	$a\Delta babab\underline{b}ba\Delta\cdots$
h	$a\Delta babab\underline{b}ba\Delta\cdots$

□

3.2.2 Schémas de combinaisons de machines de Turing

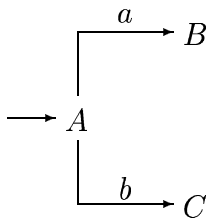
Nous introduisons maintenant une notation qui permet de décrire de manière plus compacte des combinaisons de machines de Turing.

3.11 Notation. Nous utiliserons les conventions simplificatrices suivantes, où les variables x, y et z représentent des symboles de l'alphabet du ruban et A, B, C des machines de Turing.

- $A \xrightarrow{x} B$ indique que A passe le contrôle à B si le symbole courant est x.
- $A \xrightarrow[y]{x} B$ est simplifié en $A \xrightarrow{x,y} B$.
- $A \xrightarrow{\neg x} B$ signifie que A passe le contrôle à B pour tout symbole courant différent de x.
- $A \longrightarrow B$ signifie que A passe le contrôle à B indépendamment du symbole courant.
- $\longrightarrow A \longrightarrow B \longrightarrow C$ est simplifié en $\longrightarrow ABC$.
- $\left.\xrightarrow{x,y,z}\right\} \xrightarrow{\omega}$ indique que si le symbole courant est l'un des symboles x, y, z, alors il est représenté par ω. Cette convention diminue l'encombrement dans les diagrammes. **ATTENTION :** ω ne fait pas partie de l'alphabet de la machine ; celle-ci ignore son existence. C'est une convention pour les humains.
- La machine s'arrête si x est le symbole courant et qu'il n'y a pas de transition pour x.

□

3.12 Exemple. La combinaison de machines de l'exemple 3.10 est représentée par le schéma ci-dessous.

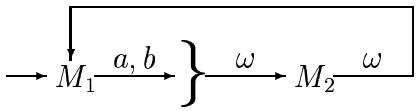


La machine A commence l'exécution (elle déplace la tête vers la droite) puis

- elle passe le contrôle à B si le nouveau symbole courant est a,
- elle passe le contrôle à C si le nouveau symbole courant est b,
- elle s'arrête si le nouveau symbole courant est Δ, car après l'exécution de A, il n'y a pas de transition pour Δ ; i.e., on considère que la machine passe dans l'état final lorsqu'il n'y a pas de transition définie dans le schéma de combinaison.

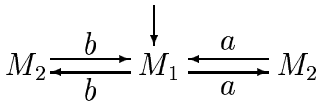
□

3.13 Exemple. Soit la machine de Turing suivante.



Ce diagramme indique que si M_2 commence l'exécution avec a ou b comme symbole courant (symbole sous la tête) et si elle termine avec le même symbole courant, alors elle passe le contrôle à M_1, sinon elle s'arrête. Si le symbole courant est Δ après l'exécution de M_1 ou de M_2, il y a arrêt.

Voici la même machine, un peu plus détaillée.

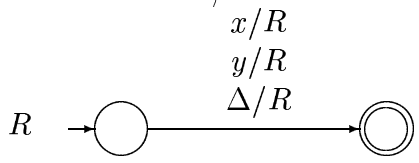


□

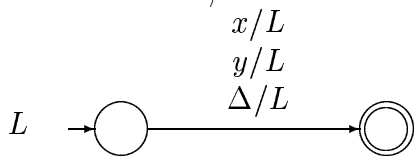
3.2.3 Blocs élémentaires

Nous allons définir une série de machines élémentaires qui vont nous servir de blocs de base pour la construction de machines plus complexes. Voyez les figures 3.5, 3.6 et 3.7 de [Brookshear89]. Nous supposerons que l'alphabet du ruban est $\Gamma = \{x, y, \Delta\}$. Il est facile de généraliser à d'autres alphabets.

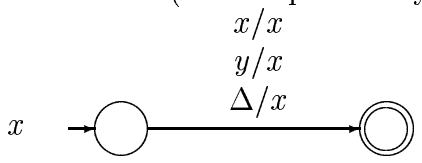
1. La machine R déplace la tête de lecture/écriture d'une case vers la droite.



2. La machine L déplace la tête de lecture/écriture d'une case vers la gauche.



3. Pour $x \in \Gamma$, la machine x écrit un x (elle remplace le symbole courant par x).



Avec les trois machines ci-dessus, nous pouvons former des machines plus complexes, comme $\rightarrow RyL$: cette machine écrit un y à droite de la position initiale de la tête, et ramène la tête dans la position initiale. Par exemple, $\rightarrow RyL$ fait passer de la configuration ruban

$$x\underline{x}x\Delta\Delta\cdots$$

à la configuration

$$x\underline{x}y\Delta\Delta\cdots$$

4. La machine R_x déplace la tête de lecture/écriture vers la droite jusqu'à ce qu'elle trouve un x. La machine $R_{\neg x}$ déplace la tête vers la droite jusqu'à ce qu'elle rencontre un symbole différent de x.

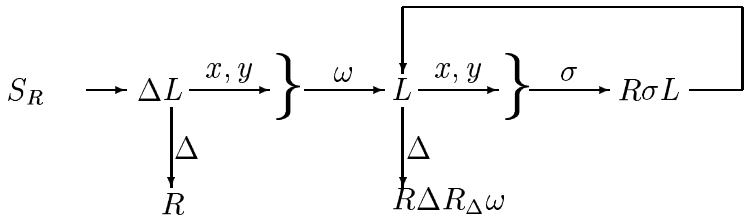
Note : les deux machines font R avant de tester le symbole. Le symbole initialement sous la tête ne compte donc pas.

De même, on construit L_x et $L_{\neg x}$, qui font la recherche vers la gauche.

Cas particuliers très utiles : $R_\Delta, R_{\neg\Delta}, L_\Delta, L_{\neg\Delta}$.

ATTENTION : Les machines R_x et Rx sont différentes (ainsi que L_x et Lx). Lorsque vous décrivez une machine de Turing, assurez-vous que votre façon d'écrire permette de distinguer ces cas. Si x est en indice, faites-le petit et écrivez-le plus bas.

5. La machine S_R (*shift right*) déplace d'une case vers la droite la séquence de symboles non blancs immédiatement à gauche de la tête dans la configuration initiale (on suppose qu'il y a un Δ à gauche de cette séquence).



Voici une suite de configurations du ruban décrivant une exécution de S_R. Il n'est pas possible de donner l'état de la machine lorsqu'on donne une configuration, puisque les descriptions schématiques comme celles de S_R ne précisent pas les états.

$\cdots \Delta xyy\underline{x}x\Delta\Delta\cdots$

$\langle$ Exécution de Δ. $\rangle$

$\cdots \Delta xyy\underline{\Delta}x\Delta\Delta\cdots$

$\langle$ Exécution de L. $\rangle$

$\cdots \Delta xy\underline{y}\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, exécution de L. $\rangle$

$\cdots \Delta x\underline{y}y\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = y$, exécution de R. $\rangle$

$\cdots \Delta xy\underline{y}\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = y$, exécution de σ. $\rangle$

$\cdots \Delta xy\underline{y}\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = y$, exécution de L. $\rangle$

$\cdots \Delta x\underline{y}y\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = y$, exécution de L. $\rangle$

$\cdots \Delta\underline{x}yy\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = x$, exécution de R. $\rangle$

$\cdots \Delta x\underline{y}y\Delta x\Delta\Delta\cdots$

$\langle$ $\omega = y$, $\sigma = x$, exécution de σ. $\rangle$

$\cdots \Delta x\underline{x}y\Delta x\Delta\Delta\cdots$

$$\langle\ \omega = y,\ \sigma = x,\ \text{exécution de } L.\ \rangle$$
$$\cdots \Delta \underline{x} x y \Delta x \Delta\Delta \cdots$$
$$\langle\ \omega = y,\ \sigma = x,\ \text{exécution de } L.\ \rangle$$
$$\cdots \underline{\Delta} x x y \Delta x \Delta\Delta \cdots$$
$$\langle\ \omega = y,\ \text{exécution de } R.\ \rangle$$
$$\cdots \Delta \underline{x} x y \Delta x \Delta\Delta \cdots$$
$$\langle\ \omega = y,\ \text{exécution de } \Delta.\ \rangle$$
$$\cdots \Delta \underline{\Delta} x y \Delta x \Delta\Delta \cdots$$
$$\langle\ \omega = y,\ \text{exécution de } R_\Delta.\ \rangle$$
$$\cdots \Delta\Delta x y \underline{\Delta} x \Delta\Delta \cdots$$
$$\langle\ \omega = y,\ \text{exécution de } \omega.\ \rangle$$
$$\cdots \Delta\Delta x y \underline{y} x \Delta\Delta \cdots$$

Avec la configuration initiale $x\underline{\Delta}\Delta\Delta\cdots$, il y a terminaison anormale (la tête tombe en bas du ruban).

Remarquez la méthode employée par S_R. Elle marque la position initiale avec un Δ qu'elle recherche à la fin. Un autre symbole spécial pourrait être utilisé au lieu de Δ.

Similairement à S_R, la machine S_L déplace d'une case vers la gauche la séquence de symboles non blancs immédiatement à droite de la tête dans la configuration initiale (on suppose qu'il y a un Δ à droite de cette séquence).

Les machines des figures 3.8, 3.9 et 3.10 de [Brookshear89] sont de bons exemples d'utilisation des machines élémentaires définies ci-dessus :

- La machine de la figure 3.8 de [Brookshear89] fait la copie d'une séquence comprise entre deux blancs. C'est-à-dire qu'elle passe d'une configuration de la forme
  $\cdots \underline{\Delta} w \Delta \cdots$
  à la configuration
  $\cdots \Delta w \underline{\Delta} w \Delta \cdots$
  où $w \in \Sigma^*$. Notez que l'écriture du w de droite peut se faire dans des cases qui à l'origine ne contiennent pas nécessairement Δ.
- La machine de la figure 3.9 de [Brookshear89] décrémente une valeur binaire de 1. Par exemple, elle passe de la configuration
  $\underline{\Delta}101\Delta\cdots$
  à la configuration
  $\underline{\Delta}100\Delta\cdots$
- La machine de la figure 3.10 de [Brookshear89] remplace la séquence w_1 par la

séquence w_2 qui la suit dans un certain ordre lexicographique. Elle passe de la configuration

$$\underline{\Delta} w_1 \Delta\Delta \cdots$$

à la configuration

$$\underline{\Delta} w_2 \Delta\Delta \cdots$$

L'alphabet de la machine est $\Sigma = \{x, y, z\}$. L'ordre lexicographique (sur Σ^*) qui est utilisé est

$$\lambda, x, y, z, xx, yx, zx, xy, yy, zy, xz, yz, zz, xxx, \ldots$$

3.14 Exercice. En utilisant les machines de Turing élémentaires présentées ci-dessus, construisez une machine qui complémente la séquence de 0 et de 1 immédiatement à droite de la tête de lecture/écriture et qui retourne la tête à sa position initiale. On suppose qu'il y a un Δ à droite de cette séquence. Prenez $\Gamma = \{0, 1, \Delta\}$.

1. Dans un premier temps, supposez que la configuration initiale du ruban soit $\underline{\Delta} w \Delta \cdots$, avec $w \in \{0,1\}^*$ (une telle supposition permet de trouver une machine simple). Donnez la suite des configurations du ruban si la configuration initiale est
   (a) $\underline{\Delta}010\Delta \cdots$
   (b) $\underline{\Delta}\Delta \cdots$

2. Dans un deuxième temps, supposez que la configuration initiale soit $u\underline{x}w\Delta \cdots$, avec $x \in \Gamma$ et $u, w \in \Gamma^*$. Donnez la suite des configurations du ruban si la configuration initiale est $01\underline{0}010\Delta \cdots$

□

3.2.4 Problèmes

1. En utilisant les machines de Turing élémentaires présentées à la section 3.2, construisez une machine de Turing qui passe de la configuration $\underline{\Delta}w\Delta\Delta\Delta\cdots$ à la configuration $\underline{\Delta}w\Delta v\Delta\Delta\Delta\cdots$, où $v, w \in \{a, b\}^*$ et v est la séquence w écrite à l'envers.
2. Le résultat de l'exécution de $\rightarrow RL$ peut-il être différent de celui de l'exécution de $\rightarrow LR$? Expliquez pourquoi.

3.3 Reconnaissance de langages

OBJECTIFS SPÉCIFIQUES

1. Classifier des machines de Turing décrites par des diagrammes de transitions. Les cas possibles sont : description incorrecte, description d'une machine déterministe, description d'une machine non déterministe. Expliquer la raison de chaque choix.
2. Classifier des descriptions formelles de machines de Turing. Les cas possibles sont : description incorrecte, description d'une machine déterministe, description d'une machine non déterministe. Expliquer la raison de chaque choix.
3. Décrire le langage accepté par une machine de Turing donnée.
4. Construire une machine de Turing acceptant un langage donné.
5. Transformer une machine de Turing donnée en une machine équivalente utilisant des conventions différentes pour l'acceptation.
6. Expliquer comment transformer une machine de Turing en une machine équivalente utilisant des conventions différentes pour l'acceptation.

3.3.1 Définition de la reconnaissance

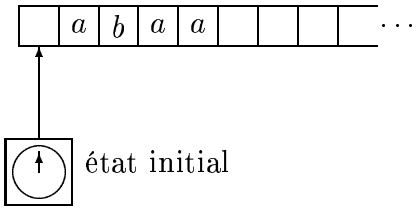
3.15 Définition. Soit L un langage sur Σ et $w \in \Sigma^*$. La séquence w est *acceptée* (ou *reconnue*) par la machine de Turing $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ si et seulement si M s'arrête normalement (dans l'état final, bien sûr) à la suite d'une exécution commençant dans la configuration

$$\iota \quad \underline{\Delta}w\Delta\Delta\cdots$$

Par exemple, avec $w = abaa$, la configuration initiale est

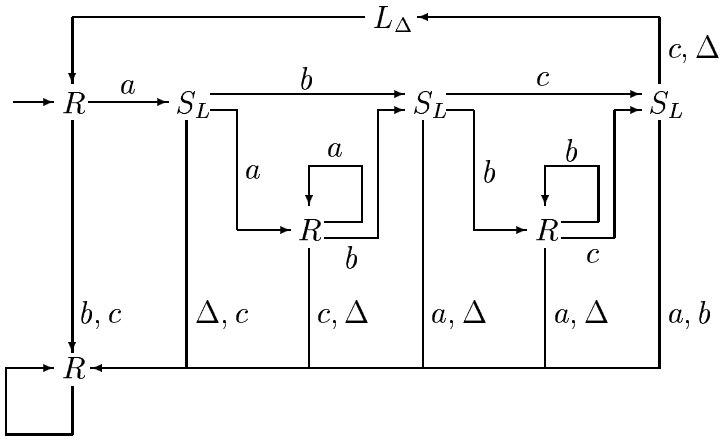
$\iota \quad \underline{\Delta}abaa\Delta\Delta\cdots$

c'est-à-dire



Comme d'habitude, $L(M) = \{w : M \text{ accepte } w\}$ est le langage *accepté* (ou *reconnu*) par M. Si $L = L(M)$ pour une machine de Turing M, on dit que L est *Turing-acceptable* ou *récursivement énumérable*. Ce dernier terme est le terme usuel ; il reflète le fait que les séquences d'un tel langage peuvent être énumérées (listées) par une machine de Turing. □

3.16 Exemple. La machine de Turing suivante accepte $\{a^n b^n c^n : n \in \mathbf{N}\}$ (on se rappelle que ce langage ne peut être accepté par un automate à pile).



Voici quelques exemples d'exécution.

1. Acceptation de $a^2b^2c^2$.

(1)	$\underline{\Delta}aabbcc\Delta\Delta\cdots$	(6)	$\Delta ab\underline{c}c\Delta\Delta\cdots$	(11)	$\Delta\underline{c}\Delta\Delta\cdots$
(2)	$\Delta\underline{a}abbcc\Delta\Delta\cdots$	(7)	$\Delta ab\underline{c}\Delta\Delta\cdots$	(12)	$\Delta\underline{\Delta}\Delta\cdots$
(3)	$\Delta\underline{a}bbcc\Delta\Delta\cdots$	(8)	$\underline{\Delta}abc\Delta\Delta\cdots$	(13)	$\underline{\Delta}\Delta\Delta\cdots$
(4)	$\Delta a\underline{b}bcc\Delta\Delta\cdots$	(9)	$\Delta\underline{a}bc\Delta\Delta\cdots$	(14)	$\Delta\underline{\Delta}\Delta\cdots$
(5)	$\Delta a\underline{b}cc\Delta\Delta\cdots$	(10)	$\Delta\underline{b}c\Delta\Delta\cdots$		

2. Acceptation de λ.

   (1) $\underline{\Delta}\Delta\Delta\cdots$
   (2) $\Delta\underline{\Delta}\Delta\cdots$

3. Rejet de a^2bc.

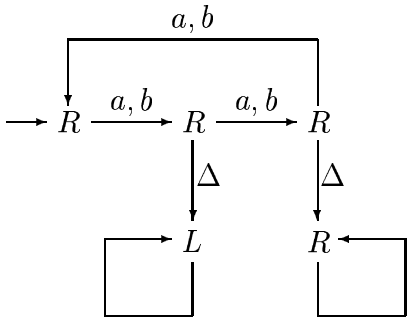
   (1) $\underline{\Delta}aabc\Delta\Delta\cdots$
   (2) $\Delta\underline{a}abc\Delta\Delta\cdots$
   (3) $\Delta\underline{a}bc\Delta\Delta\cdots$
   (4) $\Delta a\underline{b}c\Delta\Delta\cdots$
   (5) $\Delta a\underline{c}\Delta\Delta\cdots$
   (6) $\Delta a\underline{\Delta}\Delta\cdots$
   (7) $\underline{\Delta}a\Delta\Delta\cdots$
   (8) $\Delta\underline{a}\Delta\Delta\cdots$
   (9) $\Delta\underline{\Delta}\Delta\cdots$
   (10) $\Delta\Delta\underline{\Delta}\Delta\cdots$
   (11) $\Delta\Delta\Delta\underline{\Delta}\Delta\cdots$
   (12) $\Delta\Delta\Delta\Delta\underline{\Delta}\Delta\cdots$

   La machine boucle indéfiniment en déplaçant la tête vers la droite.

Remarquez que la machine ci-dessus termine toujours avec la tête sur la deuxième cellule du ruban. □

Il y a donc un langage qui n'est pas non contextuel et qui est accepté par une machine de Turing. Nous verrons plus tard que les machines de Turing peuvent accepter n'importe quel langage non contextuel. Les machines de Turing sont donc plus puissantes que les automates à pile.

3.17 Exercice. Soit $\Sigma = \{a, b\}$ l'alphabet d'entrée et $\Gamma = \{a, b, \Delta\}$ l'alphabet de ruban de la machine de Turing M suivante.



1. Dites si cette machine accepte les séquences suivantes. Pour justifier votre réponse, donnez la suite de configurations du ruban lors de l'exécution (si M s'engage dans une boucle infinie, dites-le après avoir y passé une fois).
   (a) a
   (b) ab
   (c) aba
2. Quel est $L(M)$?

□

3.18 Exercice. Construisez une machine de Turing qui accepte

$\{a^m b^n a^m b^n : m, n \in \mathbf{N}\}$.

□

3.3.2 Définition équivalente de reconnaissance

Voici une définition équivalente de reconnaissance par machine de Turing. Cette définition est plus utile pour faire certaines preuves.

3.19 Définition. *Acceptation avec message.* Soit M une machine de Turing. M *accepte* une séquence w *avec message* si

- elle s'arrête dans la configuration $\underline{\Delta}O\Delta\Delta\cdots$ lorsque $w \in L(M)$;
- elle ne s'arrête pas ou s'arrête anormalement ou s'arrête dans une configuration autre que $\underline{\Delta}O\Delta\Delta\cdots$ lorsque $w \notin L(M)$.

□

Montrons maintenant que les deux définitions d'acceptation sont équivalentes.

3.20 Théorème. *L'acceptation par arrêt (c'est-à-dire la première définition d'acceptation) est équivalente à l'acceptation avec message.*

DÉMONSTRATION.

1. Soit $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ une machine de Turing qui accepte par arrêt. Montrons qu'il existe une machine de Turing $M' = (S', \Sigma, \Gamma', \delta', \iota', h')$ qui accepte avec message et telle que $L(M) = L(M')$.

   Supposons que $\#, * \notin \Gamma$. Prenons $\Gamma' = \Gamma \cup \{\#, *\}$. Les symboles $\#, *$ seront utilisés pour délimiter la partie utilisée du ruban.

   (a) Tout d'abord, M' doit passer de la configuration $\underline{\Delta}w\Delta\Delta\cdots$ à la configuration $\#\underline{\Delta}w * \Delta\Delta\cdots$ Cela se fait grâce à la sous-machine

   $$\rightarrow R_\Delta S_R R * L_\Delta L \# R$$

   (on peut voir facilement pourquoi ça fonctionne, par exemple en prenant $\Sigma = \{a, b\}$ et la configuration $\underline{\Delta}ab\Delta\Delta\cdots$)

   (b) Ensuite, M' doit simuler M, sauf pour les deux cas suivants :

      i. Si le symbole courant est $\#$, c'est que M termine anormalement. Faire $\rightarrow L$, afin que M' termine aussi anormalement (M et M' rejettent la séquence d'entrée).

      ii. Si le symbole courant est *, autrement dit si la configuration est

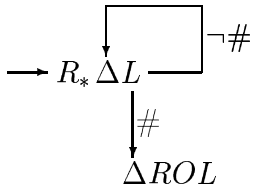
      $$\#u\underline{*}\Delta\Delta\cdots,$$

      faire $\rightarrow R * L\Delta$, pour obtenir

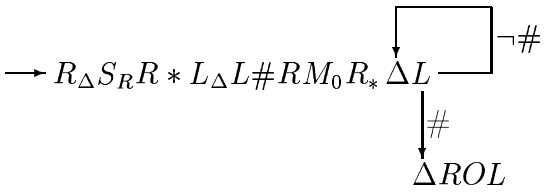
      $$\#u\underline{\Delta} * \Delta\cdots$$

      puis continuer à simuler M.

   (c) Lorsque M' est dans l'état qui correspond à l'état final de M, effacer le ruban et écrire O, en utilisant la machine de Turing suivante.



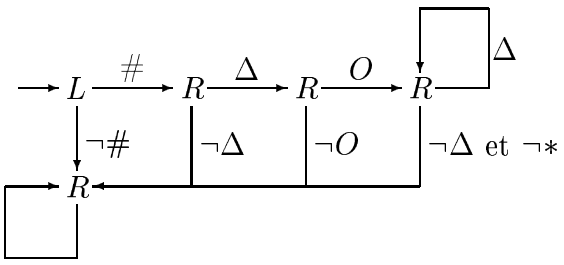
La configuration finale de cette machine est $\underline{\Delta}O\Delta\Delta\cdots$
La machine M' est donc



où M_0 est M avec les modifications décrites en (1b) ci-dessus.
(Voyez l'exemple 3.21.)

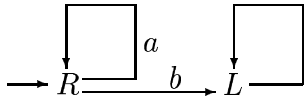
2. Soit maintenant une machine $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ qui accepte avec message. Il existe une machine de Turing $M' = (S', \Sigma, \Gamma', \delta', \iota', h')$ qui accepte par arrêt telle que $L(M) = L(M')$.
   Il suffit de faire comme en (1a) et (1b) ci-dessus, c'est-à-dire de délimiter w par $\#$ et *, puis de simuler M. Au lieu de (1c), faire ce qui suit :

   > Si M' atteint un état correspondant à l'état final de M, vérifier si la configuration est $\#\underline{\Delta}O\Delta\Delta\cdots\Delta\Delta * \Delta\Delta\cdots$ Si oui, c'est que M accepte ; en ce cas, arrêter, sinon boucler indéfiniment, afin de rejeter la séquence d'entrée. Cela se fait avec la machine suivante.

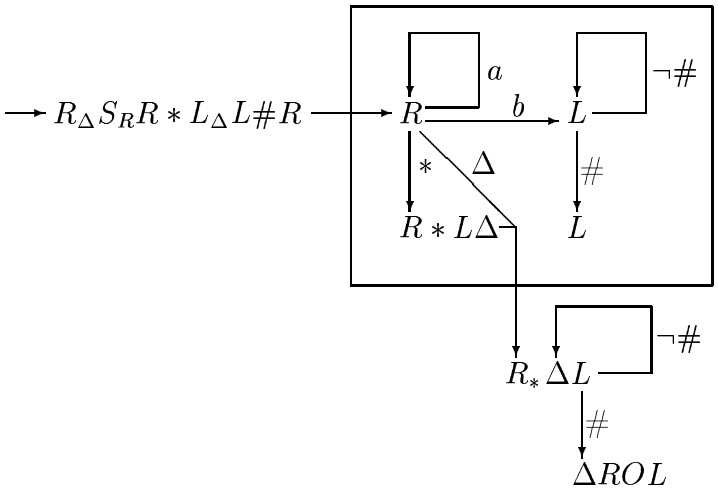


□

3.21 Exemple. Supposons que $\Gamma = \{a, b, \Delta\}$ et que M soit



La machine M accepte (par arrêt) n'importe quelle séquence de symboles a (elle s'arrête sur un Δ), mais termine anormalement si elle rencontre un b. La machine suivante accepte $L(M)$ avec message. La modification M_0 de M est la partie encadrée.



Dans la simulation M_0 de M, il est inutile de vérifier si le symbole courant est # après l'exécution de R ou s'il est $*$ après l'exécution de L, puisque cela ne peut se produire.

Voyons deux exemples d'exécution de cette machine.

$\underline{\Delta}a\Delta\Delta\cdots$

$\langle$ Exécution de R_Δ. $\rangle$

$\Delta a\underline{\Delta}\Delta\cdots$

$\langle$ Exécution de S_R. $\rangle$

$\Delta\Delta\underline{a}\Delta\cdots$

$\langle$ Exécution de $R*$. $\rangle$

$\Delta\Delta a\underline{*}\Delta\cdots$

$\langle$ Exécution de L_Δ. $\rangle$

$\Delta\underline{\Delta}a * \Delta \cdots$

$\langle$ Exécution de $L\#R$. $\rangle$

$\#\underline{\Delta}a * \Delta \cdots$

$\langle$ Exécution de R. $\rangle$

$\#\Delta\underline{a} * \Delta \cdots$

$\langle$ Exécution de R. $\rangle$

$\#\Delta a\underline{*}\Delta \cdots$

$\langle$ Exécution de $R * L\Delta$. $\rangle$

$\#\Delta a\underline{\Delta} * \Delta \cdots$

$\langle$ Exécution de R_*. $\rangle$

$\#\Delta a\Delta\underline{*}\Delta \cdots$

$\langle$ Exécution de ΔL. $\rangle$

$\#\Delta a\underline{\Delta}\Delta \cdots$

$\langle$ Exécution de ΔL. $\rangle$

$\#\Delta\underline{a}\Delta\Delta \cdots$

$\langle$ Exécution de ΔL. $\rangle$

$\#\underline{\Delta}\Delta\Delta \cdots$

$\langle$ Exécution de ΔL. $\rangle$

$\underline{\#}\Delta\Delta\Delta \cdots$

$\langle$ Exécution de ΔROL. $\rangle$

$\underline{\Delta}O\Delta\Delta \cdots$

La séquence a est acceptée avec message, comme il se doit.

$\underline{\Delta}b\Delta\Delta \cdots$

$\langle$ Exécution de $R_\Delta S_R R * L_\Delta L\#R$. $\rangle$

$\#\underline{\Delta}b * \Delta \cdots$

$\langle$ Exécution de R. $\rangle$

$\#\Delta\underline{b} * \Delta \cdots$

$\langle$ Exécution de L. $\rangle$

$\#\underline{\Delta}b * \Delta \cdots$

⟨ Exécution de L. ⟩

$\underline{\#}\Delta b * \Delta \cdots$

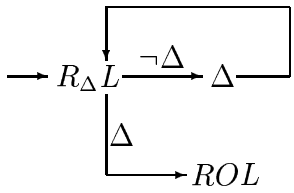
⟨ Exécution de L. ⟩

kaboum.

La séquence b est rejetée par cette machine, tout comme elle l'est par M. □

3.22 Remarque. À moins d'avis contraire, la définition d'acceptation utilisée par la suite est l'acceptation par arrêt. □

3.23 Exercice. Soit $\Sigma = \{a, b\}$ l'alphabet d'entrée et $\Gamma = \{a, b, \Delta, O\}$ l'alphabet de ruban de la machine M suivante. M accepte $\{a, b\}^*$ avec message. Utilisez la deuxième partie du théorème 3.20 pour construire une machine de Turing M' qui accepte par arrêt et telle que $L(M') = L(M)$.



□

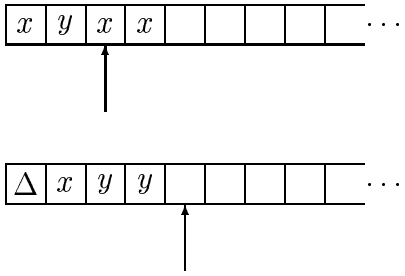
3.24 Exercice.

1. Construisez une machine de Turing qui accepte Σ^*, où $\Sigma = \{a, b\}$,
   (a) par arrêt ;
   (b) avec message.
2. Construisez une machine de Turing qui accepte $\emptyset$, avec $\Sigma = \{a, b\}$.

□

3.3.3 Machines de Turing à plusieurs rubans

Considérons une machine de Turing à 2 rubans (la notion se généralise facilement à un nombre arbitraire de rubans).



Une transition d'une telle machine dépend de l'état courant et de l'ensemble des symboles sous les têtes. Une transition consiste en

- un changement d'état et
- une écriture ou un déplacement *sur un seul ruban.*

La configuration initiale est $\underline{\Delta} w_1 \Delta\Delta\cdots$ pour le ruban 1 et $\underline{\Delta}\Delta\Delta\cdots$ pour le ruban 2.

3.25 Théorème. *Pour toute machine de Turing M à k rubans ($k \in \mathbf{N}^+$), il y a une machine de Turing M' à un ruban telle que $L(M) = L(M')$.*

DÉMONSTRATION. Donnons un aperçu de la preuve. Supposons que M a deux rubans. L'idée de la preuve est que M' utilise des quadruplets pour encoder la même information sur un ruban unique. Par exemple, les deux rubans ci-dessus seraient représentés par le ruban suivant. La ligne 1 est le contenu du ruban 1 ; la ligne 2 indique la position de la tête de lecture/écriture du ruban 1 ; la ligne 3 est le contenu du ruban 2 ; la ligne 4 indique la position de la tête du ruban 2. M' considère que chaque case de ce ruban contient un seul symbole. Le ruban contient le symbole $\#$ dans la première cellule. M' s'en sert comme marqueur afin de simuler M.

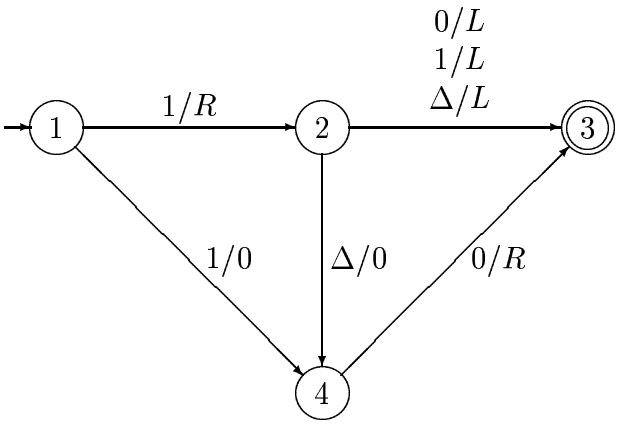
#	x	y	x	x	Δ	Δ	Δ
	Δ	Δ	1	Δ	Δ	Δ	Δ
	Δ	x	y	y	Δ	Δ	Δ
	Δ	Δ	Δ	Δ	1	Δ	Δ

□

Donc on ne gagne pas de puissance en utilisant des machines à plusieurs rubans.

3.3.4 Machines de Turing non déterministes

Soit $\Gamma = \{0, 1, \Delta\}$. Considérons la machine suivante.



Cette machine n'est pas déterministe :

1. Elle n'est pas totalement définie, car
   - il n'y a pas de transitions pour 0 et Δ dans l'état 1 ;

- il n'y a pas de transitions pour 1 et Δ dans l'état 4.

2. Elle est ambiguë, car
   - il y a deux transitions possibles si le symbole courant est 1 dans l'état 1 ;
   - il y a deux transitions possibles si le symbole courant est Δ dans l'état 2.

3.26 Définition. Tout comme une machine de Turing déterministe, une machine de Turing non déterministe est un sixtuplet $(S, \Sigma, \Gamma, \rho, \iota, h)$; cependant la quatrième composante est une relation plutôt qu'une fonction ; c'est-à-dire qu'on a

$$\rho \subseteq (S - \{h\}) \times \Gamma \times S \times (\Gamma \cup \{L, R\})$$

plutôt que

$$\delta : (S - \{h\}) \times \Gamma \to S \times (\Gamma \cup \{L, R\}).$$

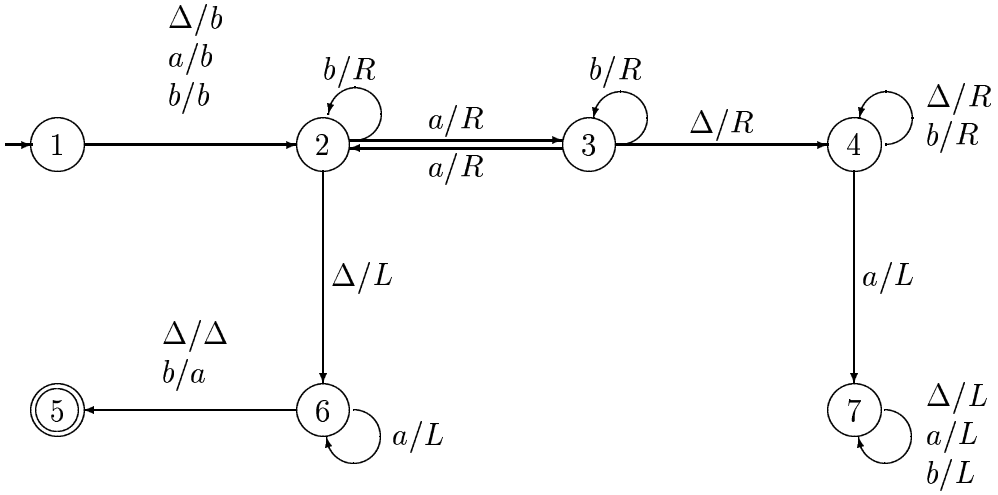
Le langage accepté par une machine de Turing non déterministe (ou *reconnu*) M est

$$L(M) = \{w : M \textit{ peut} \text{ atteindre l'état final à partir de la configuration initiale } \iota \ \underline{\Delta} w \Delta\Delta\cdots\}.$$

□

3.27 Exercice. Soient $\Sigma = \{a\}$ l'alphabet d'entrée et $\Gamma = \{a, b, \Delta\}$ l'alphabet de ruban des machines de Turing suivantes. Pour chacune d'elles, dites s'il s'agit bien d'une machine de Turing avec alphabet d'entrée Σ et alphabet de ruban Γ. Si non, dites pourquoi. Si oui, dites s'il s'agit d'une machine déterministe et dites pourquoi.

1. Machine 1.



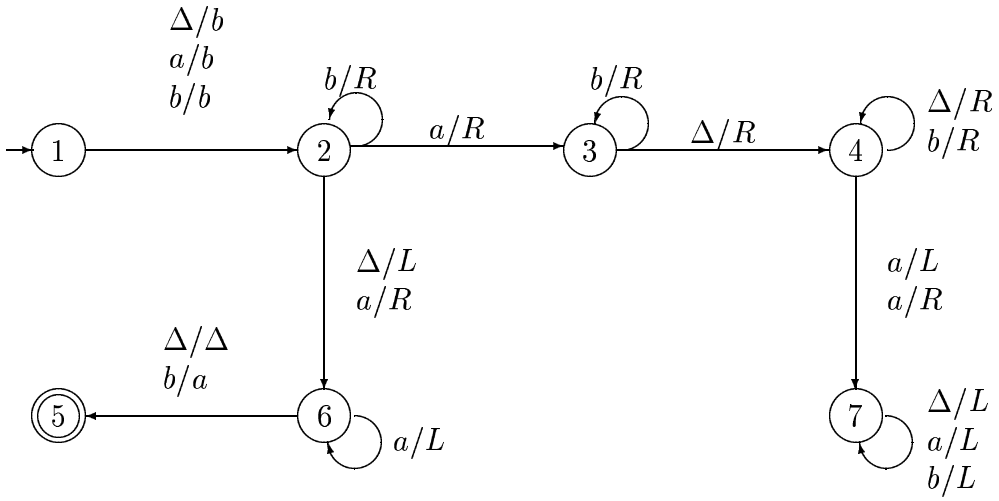
2. Machine 2.

$$(\{1,2,3,4,5,6,7\}, \{a\}, \{a,b,\Delta\}, \tau, 0, 8),$$

où τ est l'ensemble des transitions suivantes.

$(1,a,2,b)$	$(1,b,2,b)$	$(1,\Delta,2,b)$	$(2,c,3,R)$	$(2,b,2,R)$
$(2,\Delta,6,L)$	$(3,a,2,R)$	$(3,b,3,R)$	$(3,\Delta,4,R)$	$(4,a,7,L)$
$(4,b,4,G)$	$(4,\Delta,4,R)$	$(6,a,6,L)$	$(6,b,5,a)$	$(6,\Delta,5,\Delta)$
$(7,a,7,L)$	$(7,b,7,L)$	$(7,\Delta,7,L)$		

3. Machine 3.



□

3.28 Théorème. *Pour chaque machine de Turing non déterministe M, il existe une machine de Turing déterministe D telle que $L(M) = L(D)$.*

DÉMONSTRATION. Voici un aperçu de la preuve. La machine D a 3 rubans.

1. Le ruban 1 contient la séquence d'entrée w.
2. Le ruban 2 est un ruban de travail sur lequel E copie w et essaie chacune des séquences d'exécution possibles.

3. Le ruban 3 contient une séquence d'exécution (séquence de transitions). Après l'essai d'une séquence, la séquence suivante est générée en ordre lexicographique (voyez la figure 3.10 de [Brookshear89] pour une machine qui peut faire ce travail).

Par exemple, la machine de Turing non déterministe de la page 198 a sept transitions ; supposons qu'elles sont numérotées de 1 à 7. Les séquences d'exécution seraient

$\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle, \langle 5\rangle, \langle 6\rangle, \langle 7\rangle,$
$\langle 1,1\rangle, \langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 1,6\rangle, \langle 1,7\rangle,$
$\langle 2,1\rangle, \langle 2,2\rangle, \ldots \langle 1,1,1\rangle, \langle 1,1,2\rangle, \langle 1,1,3\rangle, \ldots$

Plusieurs de ces séquences de transitions ne correspondent pas à une exécution possible. Ce n'est pas grave ; ces séquences incorrectes conduisent simplement à un rejet de la séquence d'entrée. L'important c'est de générer éventuellement n'importe quelle séquence possible. Si l'une des simulations conduit à l'état d'acceptation (ici, 3), D accepte. □

Remarquez donc que l'ajout du non-déterminisme ne permet pas de reconnaître plus de langages. Cependant, on voit que l'acceptation d'une séquence w par D peut nécessiter un nombre d'étapes beaucoup plus grand que celui qui est nécessaire à M pour accepter w.

3.3.5 Problème

1. Montrez que si le langage L est acceptable par une machine de Turing, alors il existe une autre machine de Turing qui termine anormalement si et seulement si la séquence d'entrée qui lui est soumise appartient à L. (Suggestion : jetez un coup d'oeil au théorème 3.20.)

3.4 Langages Turing-acceptables

OBJECTIFS SPÉCIFIQUES

1. Décrire le langage généré par une grammaire à structures de phrase donnée.
2. Construire une grammaire à structures de phrase générant un langage donné.

Une *grammaire à structures de phrase* est une grammaire sans restrictions. Le côté gauche peut être n'importe quoi, pourvu qu'il contienne au moins un symbole non terminal. La grammaire suivante est une grammaire à structures de phrase. Le

langage généré par cette grammaire est $\{a^n b^n c^n : n \in \mathbf{N}\}$ (on se rappelle qu'aucune grammaire non contextuelle ne peut générer ce langage).

$S \to abNSc$ $\quad$ $bNa \to abN$ $\quad$ $bNb \to bbN$
$S \to \lambda$ $\quad$ $bNc \to bc$

Allons-y pour une petite dérivation.

S
$\Rightarrow$ $\quad$ $\langle$ Règle $S \to abNSc.$ $\rangle$
$abNSc$
$\Rightarrow$ $\quad$ $\langle$ Règle $S \to abNSc.$ $\rangle$
$abNabNScc$
$\Rightarrow$ $\quad$ $\langle$ Règle $S \to abNSc.$ $\rangle$
$abNabNabNSccc$
$\Rightarrow$ $\quad$ $\langle$ Règle $S \to \lambda.$ $\rangle$
$abNabNabNccc$

Quel N devrions-nous réduire ? Nous allons essayer deux possibilités (il y en a d'autres, bien sûr).

1. Dérivation à gauche.

   $\Rightarrow$ $\quad$ $\langle$ Règle $bNa \to abN.$ $\rangle$
   $aabNbNabNccc$
   $\Rightarrow$ $\quad$ $\langle$ Règle $bNb \to bbN.$ $\rangle$
   $aabbNNabNccc$
   $\Rightarrow$ $\quad$ $\langle$ Règle $bNc \to bc.$ $\rangle$
   $aabbNNabccc$

   A la dernière étape, seule la règle 4 pouvait être appliquée. Il n'est maintenant plus possible de continuer, car aucune règle n'est applicable.

2. Dérivation à droite.

   $\Rightarrow$ $\quad$ $\langle$ Règle $bNc \to bc.$ $\rangle$
   $abNabNabccc$
   $\Rightarrow$ $\quad$ $\langle$ Règle $bNa \to abN.$ $\rangle$
   $abNaabNbccc$

$\Rightarrow$ $\langle$ Règle $bNb \to bbN.$ $\rangle$

$abNaabbNccc$

$\Rightarrow$ $\langle$ Règle $bNc \to bc.$ $\rangle$

$abNaabbccc$

$\Rightarrow$ $\langle$ Règle $bNa \to abN.$ $\rangle$

$aabNabbccc$

$\Rightarrow$ $\langle$ Règle $bNa \to abN.$ $\rangle$

$aaabNbbccc$

$\Rightarrow$ $\langle$ Règle $bNb \to bbN.$ $\rangle$

$aaabbNbccc$

$\Rightarrow$ $\langle$ Règle $bNb \to bbN.$ $\rangle$

$aaabbbNccc$

$\Rightarrow$ $\langle$ Règle $bNc \to bc.$ $\rangle$

$aaabbbccc$

Donc, contrairement à ce qui se passe avec les grammaires non contextuelles, l'ordre de dérivation est important.

Dans les dérivations ci-dessus, remarquez bien comment le symbole N « traîne » le b jusqu'au c pour finalement se suicider. C'est un truc à retenir.

3.29 Théorème. *Les langages reconnus par les machines de Turing sont exactement les langages générés par les grammaires à structures de phrase.* □

3.30 Exercice. Trouvez une grammaire à structures de phrase qui génère le langage $\{a^n b^{2n} c^{4n} : n \in \mathbf{N}\}$.

□

3.31 Théorème. *Pour tout alphabet Σ il y a un langage qui n'est pas un langage à structures de phrase (c'est-à-dire qu'il ne peut être accepté par une machine de Turing).*

DÉMONSTRATION. Ce qui suit est un simple aperçu de la preuve. Soit L un langage à structures de phrase sur Σ.

1. Pour toute machine de Turing $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ telle que $L(M) = L$, il y a une machine de Turing $M' = (S', \Sigma, \Gamma', \delta', \iota', h')$ telle que $L(M') = L$ et $\Gamma' = \Sigma \cup \{\Delta\}$. *Par exemple*, si

   $\Sigma = \{a, b\}$
   $\Gamma = \{a, b, \Delta, \#\}$
   $\Gamma' = \{a, b, \Delta\}$,

   on utilise l'encodage défini par

   $a \to a$
   $b \to aa$
   $\# \to aaa$
   $\Delta \to \lambda$

   (En fait, le symbole b n'est même pas utilisé par M'.) Le contenu du ruban de M est encodé en codant chaque symbole selon ce codage et en séparant chaque symbole par un Δ. La séquence

   $$a\Delta b\#\Delta$$

   est donc encodée par

   $$\Delta a\Delta\Delta aa\Delta aaa\Delta\Delta.$$

   Donc, pour tout langage à structures de phrase L sur Σ, il existe une machine de Turing M avec alphabet de pile $\Sigma \cup \{\Delta\}$ telle que $L(M) = L$.

2. L'ensemble des langages sur Σ (c'est-à-dire $\wp(\Sigma^*)$) est non dénombrable. Or on peut énumérer les machines de Turing ayant un alphabet de pile $\Gamma = \Sigma \cup \{\Delta\}$. On énumère d'abord les machines ayant 2 états, puis celles à 3 états, ... Il y a donc plus de langages que de machines de Turing.

□

Une question intéressante qu'on peut alors se poser est : Le langage naturel est-il reconnaissable par une machine de Turing ?

3.4.1 Problème

1. Construisez une grammaire à structures de phrase G telle que

   $$L(G) = \{a^m b^n a^m b^n : m, n \in \mathbf{N}\}.$$

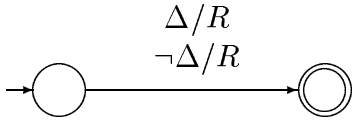
3.5 Au delà des langages à structures de phrase

OBJECTIFS SPÉCIFIQUES

1. Construire une machine de Turing décidant un langage donné.
2. Déterminer si un langage donné est décidable (récursif).
3. Déterminer si un langage donné est un langage à structures de phrase (est récursivement énumérable).

3.5.1 Encodage des machines de Turing

On peut encoder la description d'une machine de Turing sur un alphabet Σ au moyen de zéros (0) et de uns (1), c'est-à-dire qu'on peut représenter une machine de Turing par un nombre naturel en notation binaire ([Brookshear89, pages 176–178]). Certains nombres ne correspondent à aucune machine. Associons-leur la machine suivante.



3.5.2 Un langage qui n'est pas un langage à structures de phrase

L'encodage ci-dessus nous donne une *fonction surjective* $f : \mathbf{N} \to \{$ machines de Turing $\}$, pour un alphabet Σ donné.

Soit $w \in \Sigma^*$. Notons M_w la machine $f(|w|)$: c'est la machine avec alphabet Σ dont l'encodage est le nombre *longueur de* w (si $|w|$ n'est pas l'encodage d'une machine de Turing, prendre la machine ci-dessus pour M_w).

Soit maintenant le langage

$$L_0 = \{w : M_w \text{ n'accepte pas } w\}.$$

3.32 Théorème. *L_0 n'est pas Turing-acceptable.*

DÉMONSTRATION. La preuve est une preuve par contradiction. *Supposons au contraire qu'il existe* une machine de Turing M qui accepte L_0. Par la partie 1 du théorème 3.31, on peut supposer que l'alphabet de ruban de M est $\Sigma \cup \{\Delta\}$. La machine M peut alors être encodée, selon la technique de [Brookshear89], par un nombre binaire, disons n. Il existe certainement une séquence $w_0 \in \Sigma^*$ telle que $|w_0| = n$. La machine M reconnaissant L_0 est alors M_{w_0}, et on a $L_0 = L(M_{w_0})$.

Question : A-t-on $w_0 \in L(M_{w_0})$?

1. Supposons que oui. Puisque $L_0 = L(M_{w_0})$, on a $w_0 \in L_0$. Par définition de L_0, cela veut dire que M_{w_0} n'accepte pas w_0, et donc que $w_0 \notin L(M_{w_0})$. C'est une contradiction ; on ne peut donc avoir $w_0 \in L(M_{w_0})$. Examinons l'autre possibilité.
2. Supposons que non (c'est-à-dire supposons que $w_0 \notin L(M_{w_0})$). Puisque $L_0 = L(M_{w_0})$, on a $w_0 \notin L_0$. Par définition de L_0, cela veut dire que M_{w_0} accepte w_0, et donc que $w_0 \in L(M_{w_0})$. C'est une contradiction ; on ne peut donc avoir $w_0 \notin L(M_{w_0})$.

Comme nous ne pouvons avoir ni $w_0 \in L(M_{w_0})$ ni $w_0 \notin L(M_{w_0})$, cela nous force à conclure que la supposition initiale est erronée. Il n'existe donc pas de machine de Turing qui puisse reconnaître L_0. □

Cela veut dire qu'on ne peut construire une machine de Turing qui analyse les autres machines de Turing et qui termine si et seulement si la machine analysée ne termine pas.

3.5.3 Machines de Turing universelles

Ce sont des machines « programmables ». On leur donne en entrée la description encodée d'une machine de Turing M et d'une séquence w. La machine de Turing universelle peut alors simuler l'exécution de M sur l'entrée w et accepter si et seulement si M accepte w. [Brookshear89, figure 3.24] présente une machine de Turing universelle à 3 rubans. Par le théorème 3.25, il existe donc une machine de Turing universelle à un ruban.

Soit T_u une machine de Turing universelle à un ruban. Soit M_{pre} la machine de Turing qui transforme w en la représentation binaire de $|w|w$ (en d'autres termes, M_{pre} génère l'encodage de M_w à partir de w, puis place l'encodage de w à sa suite). Si $|w|$ n'est pas une machine de Turing, prendre l'encodage de la machine de la page 205.

Que fait la machine $\rightarrow M_{pre} \rightarrow T_u$? Elle reconnaît le langage

$$L_1 = \{w : M_w \text{ accepte } w\}$$

car $\rightarrow M_{pre} \rightarrow T_u$ s'arrête si et seulement si M_w s'arrête sur w. L_1 est donc le complément de L_0.

Il est intéressant de voir que même si un langage n'est pas reconnaissable, son complément peut l'être.

3.5.4 Langages acceptables versus langages décidables

3.33 Définition. Un langage L est *décidable* (ou *récursif*) si et seulement s'il existe une machine de Turing M telle que

- si $w \in L$, alors M passe de la configuration initiale $\underline{\Delta}w\Delta\Delta\cdots$ à la configuration finale $\underline{\Delta}O\Delta\Delta\cdots$
- si $w \notin L$, alors M passe de la configuration initiale $\underline{\Delta}w\Delta\Delta\cdots$ à la configuration finale $\underline{\Delta}N\Delta\Delta\cdots$

C'est-à-dire que M s'arrête toujours en répondant *oui* ou *non* selon que $w \in L$ ou $w \notin L$ (on dit que M *décide* L). □

3.34 Exemple. Le langage $\{a^n b^n c^n : n \in \mathbf{N}\}$ est décidable, par le problème 1, page 210.

Tous les langages non contextuels sont décidables [Brookshear89, pages 185–187]. Un compilateur peut donc toujours analyser une séquence et déterminer (i.e. répondre *oui* ou *non*) si c'est une structure non contextuelle). □

3.5.5 Le problème de l'arrêt

La description d'une machine de Turing peut être encodée au moyen d'une séquence de 0 et de 1 [Brookshear89, pages 176–178]. Soit $\rho(M)$ l'encodage binaire d'une machine de Turing M. Considérons les machines de Turing avec $\Sigma = \{0, 1\}$ et $\Gamma = \{0, 1, \Delta\}$. Que se passe-t-il si l'on donne $\rho(M)$ en entrée à M ? Ou bien M s'arrête ou bien elle ne s'arrête pas (pas très fort comme résultat). Soit maintenant

$$L_h = \{\rho(M) : M \text{ accepte } \rho(M)\};$$

L_h contient les descriptions des machines de Turing qui s'arrêtent normalement lorsque leur séquence d'entrée est $\rho(M)$.

Question : Est-ce que L_h est décidable ? A cause de la définition de L_h, on appelle ce problème le *problème de l'arrêt.*

3.35 Théorème. *L_h n'est pas décidable.*

DÉMONSTRATION. Montrons-le par contradiction. *Supposons qu'il existe* une machine de Turing M_h qui décide L_h. Modifions M_h pour qu'elle réponde par

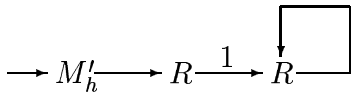
$$\underline{\Delta}1\Delta\Delta\cdots \text{ et } \underline{\Delta}0\Delta\Delta\cdots$$

au lieu de

$$\underline{\Delta}O\Delta\Delta\cdots \text{ et } \underline{\Delta}N\Delta\Delta\cdots$$

respectivement. Appelons cette machine modifiée M_h'. L'alphabet de ruban de M_h' est $\{0, 1, \Delta\}$.

Soit la machine M_0 avec alphabet de ruban $\Gamma_0 = \{0, 1, \Delta\}$ définie ainsi :



M_0 arrête
si et seulement si M'_h arrête dans la configuration $\underline{\Delta}0\Delta\Delta\cdots$
si et seulement si M_h arrête dans la configuration $\underline{\Delta}N\Delta\Delta\cdots$

Est-ce que M_0 s'arrête avec l'entrée $\rho(M_0)$?

- Supposons que oui. Alors, par définition de L_h, $\rho(M_0) \in L_h$, d'où M_h s'arrête avec un O, d'où M'_h s'arrête avec un 1, d'où M_0 ne s'arrête pas. Contradiction. Examinons l'autre possibilité.
- Supposons que non. Alors, $\rho(M_0) \notin L_h$, d'où M_h s'arrête avec N, d'où M'_h s'arrête avec un 0, d'où M_0 s'arrête. Contradiction.

Puisque M_0 ne peut ni s'arrêter ni ne pas s'arrêter, il faut abandonner l'hypothèse initiale qu'il existe une machine de Turing M_h décidant L_h. □

Ce théorème a une conséquence pratique très importante : Aucun algorithme ne peut analyser un programme et ses données d'entrée et terminer en disant si ce programme termine pour ces données. Le mieux que l'algorithme peut faire, c'est de simuler le programme. Mais si le programme ne s'arrête pas, l'algorithme ne s'arrête pas non plus (dans ce cas, on ne peut parler d'algorithme). Bien sûr, il existe des algorithmes pour faire l'analyse de certaines classes de programmes. Mais il n'existe pas d'algorithme pouvant analyser n'importe quel programme.

3.36 Remarque. On peut faire le même truc avec un programme. Supposons qu'on a un programme P auquel on donne en entrée des textes de programmes. Supposons que P permette de décider si un autre programme Q s'arrête lorsque l'entrée de Q est le texte de Q ; c'est-à-dire que P termine en disant *oui* si Q termine et en disant *non* si Q ne termine pas. Appelons P' le programme P modifié de sorte que P' termine avec un *non* si Q ne termine pas et boucle si Q termine (cela est facile à faire : il suffit d'ajouter une boucle infinie aux endroits où P répond *oui*). Demandons-nous maintenant si P' termine lorsqu'on lui donne en entrée le texte de P'. □

3.37 Remarque. Il existe une machine de Turing qui *accepte* L_h. Il suffit de simuler l'exécution de M sur $\rho(M)$ et d'accepter si M accepte. □

3.38 Exercice. Construisez une machine de Turing qui décide le langage $\{a^m b^n a^m b^n : m, n \in \mathbf{N}\}$.

□

3.39 Exercice. Identifiez le paradoxe dans l'affirmation suivante :

Le cuisinier d'une communauté isolée cuisine pour tous ceux et seulement ceux qui ne se font pas la cuisine.

□

3.5.6 Problème

1. Soit $L = \{a^n b^n c^n : n \in \mathbf{N}\}$. Construisez une machine de Turing M qui décide L.

3.6 Conclusion

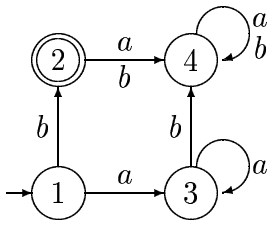
Nous avons donc la hiérarchie de langages suivante (avec des exemples), ordonnée selon la complexité des langages.

1. Réguliers : $\{a^n : n \in \mathbf{N}\}$.
2. Non contextuels déterministes : $\{a^n b^n : n \in \mathbf{N}\}$.
3. Non contextuels : $\{a^n b^n : n \in \mathbf{N}\} \cup \{a^n b^{2n} : n \in \mathbf{N}\}$.
4. Décidables : $\{a^n b^n c^n : n \in \mathbf{N}\}$.
5. A structures de phrase (Turing acceptables, ou récursivement énumérables) : L_1 (qui est $\Sigma^* - L_0$).
6. Généraux : L_0.

Chapitre 4

Solution de quelques exercices

Exercice 1.45, page 45. Soit $\Sigma = \{a, b\}$. Donnez l'automate $(S, \Sigma, \delta, \iota, F)$ correspondant au diagramme suivant :



Solution. Le diagramme est complètement défini et non ambigu. Il a donc un automate associé :

$$\begin{array}{rcll} S & = & \{1,2,3,4\} & \delta(1,a)=3 \quad \delta(1,b)=2 \\ \Sigma & = & \{a,b\} & \delta(2,a)=4 \quad \delta(2,b)=4 \\ \iota & = & 1 & \delta(3,a)=3 \quad \delta(3,b)=4 \\ F & = & \{2\} & \delta(4,a)=4 \quad \delta(4,b)=4 \end{array}$$

Voici deux autres représentations de δ :

- $\delta = \{((1,a),3),((2,a),4),((3,a),3),((4,a),4),$
  $((1,b),2),((2,b),4),((3,b),4),((4,b),4)\}$
- $\delta =$

	a	b
1	3	2
2	4	4
3	3	4
4	4	4

Exercice 1.65, page 59. Montrez que les langages suivants ne sont pas réguliers.

1. $\{a^{n!} : n \in \mathbf{N}\}$
2. $\{a^n : n \in \mathbf{N} \wedge n$ n'est pas un carré$\}$ (Indice : regardez l'exercice 1.58 et l'exemple 1.64.)

Solution.

1. La preuve procède par contradiction. Posons $L = \{a^{n!} : n \in \mathbf{N}\}$.

   L régulier

   $\Leftrightarrow$ $\langle$ Définition 1.57. $\rangle$

   $\exists M : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L$

   $\Rightarrow$ $\langle$ L contient toutes les séquences de la forme $a^{k!}$, en particulier celles pour lesquelles $k > |S|$. $\rangle$

   $\exists M : \exists k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ k > |S| \ \wedge \ \boxed{a^{k!}} \in L$

   $\Rightarrow$ $\langle$ Puisqu'il y a plus de $|S|$ symboles a à lire (car $k! > |S|$), il doit y avoir une boucle de longueur $j > 0$ et $j \leq |S|$ lors de la lecture de $a^{k!}$. Cette boucle peut être traversée $\boxed{1}$ fois de plus, d'où $a^{k!+j} \in L(M)$, puisque $a^{k!} \in L = L(M)$. $\rangle$

   $\exists M : \exists j, k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L$
   $\wedge \ 0 < j \leq |S| < k \ \wedge \ \boxed{a^{k!+j}} \in L(M)$

   $\Rightarrow$ $\langle$ $\boxed{\begin{array}{l} k! < k! + j < k! + k \leq k! + k! = 2k! \leq (k+1)k! = (k+1)!. \\ \text{Puisque } k! + j \text{ est strictement coincé entre deux factorielles} \\ \text{consécutives, ce n'est pas une factorielle, d'où } a^{k!+j} \notin L. \end{array}}$ $\rangle$

   $\exists M : \exists j, k \in \mathbf{N} : L(M) = L \ \wedge \ \boxed{a^{k!+j}} \in L(M) \ \wedge \ \boxed{a^{k!+j}} \notin L$

   $\Rightarrow$ $\langle$ $\boxed{a^{k!+j}} \in L(M) \ \wedge \ \boxed{a^{k!+j}} \notin L \ \Rightarrow \ L(M) \neq L$ $\rangle$

   $\exists M : L(M) = L \ \wedge \ L(M) \neq L$

   $\Leftrightarrow$ $\langle$ $(p \wedge \neg p) \Leftrightarrow$ faux et $(\exists x : \text{faux}) \Leftrightarrow$ faux. $\rangle$

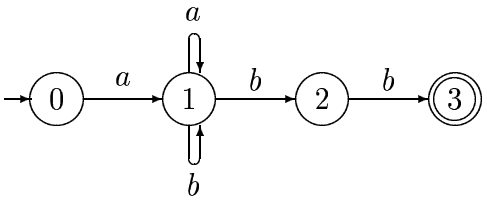
   faux.

2. La preuve de cette partie est aussi une preuve par contradiction. Posons $L = \{a^n : n \in \mathbf{N} \wedge n \text{ n'est pas un carré}\}$.

   L régulier
   $\Leftrightarrow$ $\langle$ Exercice 1.58, partie 2. $\rangle$
   Complément de L régulier
   $\Leftrightarrow$ $\langle$ Définition de L. $\rangle$
   $\{a^{n^2} : n \in \mathbf{N}\}$ est régulier
   $\Rightarrow$ $\langle$ Par l'exemple 1.64, $\{a^{n^2} : n \in \mathbf{N}\}$ n'est pas régulier. $\rangle$
   $\{a^{n^2} : n \in \mathbf{N}\}$ est régulier et n'est pas régulier
   $\Leftrightarrow$ faux

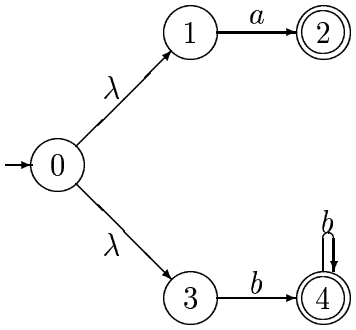
Exercice 1.69, page 64. Classifiez les automates finis ci-dessous décrits par des diagrammes de transitions sur l'alphabet Σ. Dites tout d'abord si la description est correcte. Si oui, dites si l'automate est déterministe. Expliquez la raison de chaque choix.

1. $\Sigma = \{a, b\}$



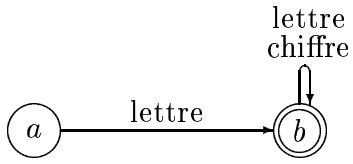
   Solution. La description est correcte. L'automate n'est pas déterministe, car il n'est pas totalement défini (par exemple, il n'y a pas de transition qui quitte l'état 3) et il est ambigu (il y a deux transitions b qui quittent l'état 2). Notez qu'une seule de ces raisons suffit.

2. $\Sigma = \{a, b\}$



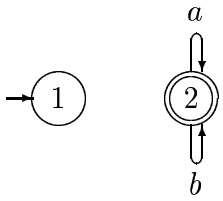
   Solution. Le diagramme est incorrect. Comme $\lambda \notin \Sigma$, λ ne peut servir à étiqueter des transitions.

3. $\Sigma =$ lettre $\cup$ chiffre



Solution. La description est incorrecte : l'état initial n'est pas indiqué.

4. $\Sigma = \{a, b\}$

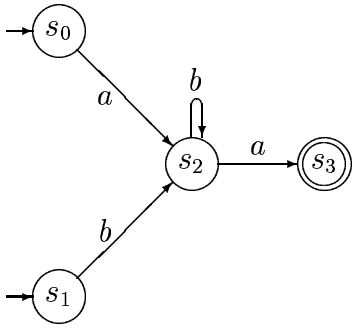


Solution. La description est correcte. L'automate n'est pas déterministe, car il n'est pas totalement défini (aucune transition ne quitte l'état 1).

5. $\Sigma = \emptyset$

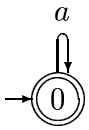
Solution. La description est incorrecte. Un alphabet ne peut être vide (voyez la définition 1.1).

6. $\Sigma = \{a, b\}$



Solution. La description est incorrecte. Un automate ne peut avoir deux états initiaux.

7. $\Sigma = \{a\}$

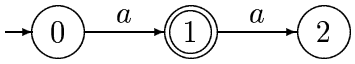


Solution. La description est correcte. L'automate est déterministe. Il y a exactement une transition par état par symbole de l'alphabet.

8. $\Sigma = \{a\}$

Solution. La description est correcte. L'automate n'est pas déterministe, car il n'est pas complètement défini (aucune transition ne quitte l'état s_0).

9. $\Sigma = \{a\}$



Solution. La description est correcte. L'automate n'est pas déterministe, car il n'est pas complètement défini (aucune transition ne quitte l'état 2).

Exercice 1.70, page 65. Classifiez les descriptions formelles suivantes d'automates finis. Dites tout d'abord si la description est correcte. Si oui, dites si l'automate est déterministe. Expliquez la raison de chaque choix.

1. $M_1 = (\{s_0, s_1, s_2\}, \{a, b\}, \delta, s_0, \{s_1\})$, où

$$\delta(s_0, a) = s_1 \quad \delta(s_0, b) = s_2$$
$$\delta(s_1, a) = s_1 \quad \delta(s_1, b) = s_2$$
$$\delta(s_2, a) = s_2 \quad \delta(s_2, b) = s_2$$

Solution. C'est bien un automate fini. L'ensemble des états est $\{s_0, s_1, s_2\}$, l'alphabet est $\{a, b\}$, δ est une fonction qui décrit les transitions en assignant un état destination à chaque combinaison possible (état source, symbole lu), s_0 est l'état initial et $\{s_1\}$ est l'ensemble des états finaux (ici, il n'y a qu'un état final). Comme δ est une fonction, l'automate est déterministe.

2. $M_2 = (\{0\}, \Sigma, \delta, \{0\})$, où

$$\Sigma = \{a, b, c\}$$
$$\delta(0, a) = 0 \quad \delta(0, b) = 0 \quad \delta(0, c) = 0$$

Solution. Ce n'est pas la description d'un automate, car l'état initial n'est pas indiqué.

3. $M_3 = (\{a, b\}, \{1, 2\}, \rho, a, \emptyset)$, où

$$\rho(a, 1) = a \quad \rho(a, 2) = b$$
$$\rho(b, 1) = a \quad \rho(b, 2) = b$$

Solution. C'est la description d'un automate fini. L'ensemble des états est $\{a, b\}$, l'alphabet est $\{1, 2\}$, ρ est une fonction qui décrit les transitions en assignant un état destination à chaque combinaison possible (état source, symbole

lu), a est l'état initial et $\emptyset$ est l'ensemble des états finaux (ici, il n'y en a aucun). Comme ρ est une fonction, l'automate est déterministe. Remarquez qu'il ne faut pas se fier à l'identificateur utilisé pour nommer la fonction ou la relation (ici, ρ désigne une fonction).

4. $M_4 = (\{s_1, s_2, s_3, s_4\}, \{a, b\}, \rho, s_1, \{s_2, s_4\})$, où

$$\begin{aligned}\rho = \{&(s_1, a, s_2), (s_1, b, s_2)\\ &(s_2, a, s_3), (s_2, b, s_4)\\ &(s_3, a, s_4), (s_3, b, s_2)\}\end{aligned}$$

Solution. C'est la description d'un automate fini. L'ensemble des états est $\{s_1, s_2, s_3, s_4\}$, l'alphabet est $\{a, b\}$, ρ est une relation de transition incluse dans $\{s_1, s_2, s_3, s_4\} \times \{a, b\} \times \{s_1, s_2, s_3, s_4\}$, s_1 est l'état initial et $\{s_2, s_4\}$ est l'ensemble des états finaux. La relation ρ ne contient pas de triplet dont la première composante est s_4 (il n'y a pas de transition qui origine de s_4) ; par conséquent, ρ n'est pas une fonction et l'automate n'est pas déterministe.

5. $M_5 = (\{1, 2, 3, 4\}, \{a, b, c\}, \delta, 1, \{2\})$, où

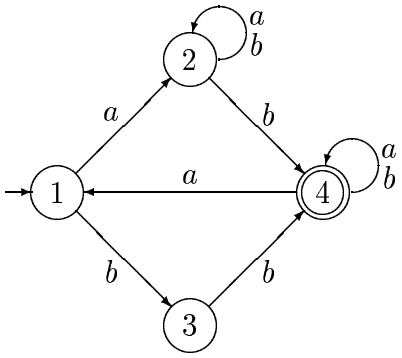
$$\begin{aligned}\delta = \{&(1, a, 2), (1, b, 3), (1, c, 3)\\ &(2, a, 2), (2, b, 4), (2, c, 4)\\ &(2, a, 3), (2, b, 3), (2, c, 3)\\ &(3, a, 3), (3, b, 3), (3, c, 3)\\ &(4, a, 3), (4, b, 3), (4, c, 3)\}\end{aligned}$$

Solution. C'est la description d'un automate fini. L'ensemble des états est $\{1, 2, 3, 4\}$, l'alphabet est $\{a, b, c\}$, δ est une relation de transition incluse dans $\{1, 2, 3, 4\} \times \{a, b, c\} \times \{1, 2, 3, 4\}$, 1 est l'état initial et $\{2\}$ est l'ensemble des états finaux (il n'y a qu'un état final). La relation δ contient les triplets $(2, a, 2)$ et $(2, a, 3)$, qui décrivent deux transitions différentes lors de la lecture d'un a dans l'état 2 ; par conséquent, δ n'est pas une fonction et l'automate n'est pas déterministe. Encore une fois, remarquez qu'il ne faut pas se fier à l'identificateur utilisé pour nommer la fonction ou la relation (ici, δ désigne une relation qui n'est pas une fonction).

6. $M_6 = (\{a, b\}, \{1\}, \emptyset, a, \{b\})$

Solution. C'est la description d'un automate fini. L'ensemble des états est $\{a, b\}$, l'alphabet est $\{1\}$, $\emptyset$ est une relation de transition, incluse dans $\{a, b\} \times \{1\} \times \{a, b\}$, a est l'état initial et $\{b\}$ est l'ensemble des états finaux (il n'y a qu'un état final). La relation $\emptyset$ n'est pas totale et n'est donc pas une fonction ; par conséquent, l'automate n'est pas déterministe.

Exercice 1.77, page 75. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un automate fini déterministe qui accepte les mêmes séquences que l'automate fini non déterministe ci-dessous. L'alphabet est $\{a, b\}$.



Solution. Le quintuplet correspondant à l'automate ci-dessus est $M = (S, \Sigma, \rho, \iota, F)$, où

- $S = \{1, 2, 3, 4\}$,
- $\Sigma = \{a, b\}$,
- $\rho = \{(1, a, 2), (1, b, 3), (2, a, 2), (2, b, 2),$
  $(2, b, 4), (3, b, 4), (4, a, 1), (4, a, 4), (4, b, 4)\}$,
- $\iota = 1$,
- $F = \{4\}$.

En suivant strictement la méthode du théorème 1.74, il faudrait construire un automate avec 16 états (car $|\wp(S)| = 2^4 = 16$). Nous pourrions ensuite éliminer les états superflus (non accessibles à partir de l'état initial). Au lieu de cela, nous allons construire l'automate déterministe demandé en générant seulement les états accessibles. Tel que prescrit par le théorème 1.74, l'état initial de cet automate déterministe est $\iota' = \{\iota\} = \{1\}$. Les transitions qui quittent cet état sont données par la table suivante, qui définit $\delta(\{1\}, a)$ et $\delta(\{1\}, b)$:

	a	b
$\{1\}$	$\{2\}$	$\{3\}$

Ces résultats sont obtenus en appliquant la définition de δ donnée dans le théorème 1.74. Par exemple,

$$
\begin{aligned}
& \delta(\{1\}, a) \\
= & \qquad \langle \text{ D'après la construction décrite dans le théorème 1.74, la fonction } \\
& \qquad \delta \text{ est définie par } \delta(s', x) = \{s \in S : \exists r \in s' : (r, x, s) \in \rho\}. \rangle \\
& \{s \in S : \exists r \in \{1\} : (r, a, s) \in \rho\} \\
= & \qquad \langle \text{ Signification du quantificateur existentiel. } \rangle
\end{aligned}
$$

$$
\begin{array}{ll}
 & \{s \in S : (1, a, s) \in \rho\} \\
= & \qquad \langle \text{ Définition de } \rho \text{ ci-dessus. } \rangle \\
 & \{2\}
\end{array}
$$

Intuitivement, ce résultat découle du fait que seul l'état 2 est accessible à partir de l'état 1 à la lecture d'un a.

Ceci nous donne deux nouveaux états, soit $\{2\}$ et $\{3\}$. Les transitions qui quittent ces états sont donnés par la table suivante :

	a	b
$\{2\}$	$\{2\}$	$\{2,4\}$
$\{3\}$	$\emptyset$	$\{4\}$

Ceci nous donne trois nouveaux états, $\emptyset, \{4\}, \{2,4\}$. La table suivante décrit les transitions qui quittent ces états :

	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{4\}$	$\{1,4\}$	$\{4\}$
$\{2,4\}$	$\{1,2,4\}$	$\{2,4\}$

Comme dernier exemple de calcul formel basé sur la définition de δ donnée dans le théorème 1.74, montrons comment $\delta(\{2,4\}, a) = \{1,2,4\}$ a été obtenu.

$$
\begin{array}{ll}
 & \delta(\{2,4\}, a) \\
= & \qquad \langle \text{ D'après la construction décrite dans le théorème 1.74, la fonction} \\
 & \qquad \delta \text{ est définie par } \delta(s', x) = \{s \in S : \exists r \in s' : (r, x, s) \in \rho\}. \rangle \\
 & \{s \in S : \exists r \in \{2,4\} : (r, a, s) \in \rho\} \\
= & \qquad \langle \text{ Signification du quantificateur existentiel. } \rangle \\
 & \{s \in S : (2, a, s) \in \rho \vee (4, a, s) \in \rho\} \\
= & \qquad \langle \text{ Définition de } \rho \text{ ci-dessus. } \rangle \\
 & \{1,2,4\}
\end{array}
$$

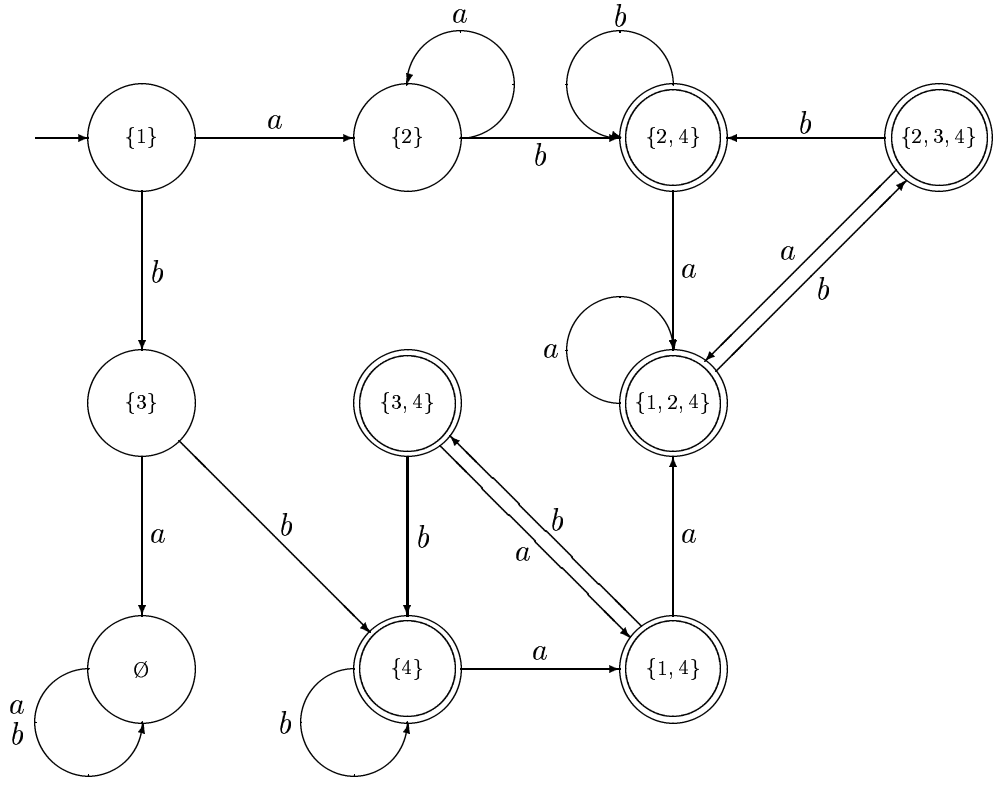
Intuitivement, ce résultat découle du fait que l'état 2 est accessible à partir de l'état 2 à la lecture d'un a et que les états 1 et 4 sont accessibles à partir de l'état 4 à la lecture d'un a.

La procédure se continue ainsi jusqu'à ce qu'aucun nouvel état ne soit généré. La table suivante résume la construction des états. La colonne “Étape" indique à quelle étape une ligne a été construite.

	a	b	Étape
$\{1\}$	$\{2\}$	$\{3\}$	1
$\{2\}$	$\{2\}$	$\{2,4\}$	2
$\{3\}$	$\emptyset$	$\{4\}$	2
$\emptyset$	$\emptyset$	$\emptyset$	3
$\{4\}$	$\{1,4\}$	$\{4\}$	3
$\{2,4\}$	$\{1,2,4\}$	$\{2,4\}$	3
$\{1,4\}$	$\{1,2,4\}$	$\{3,4\}$	4
$\{1,2,4\}$	$\{1,2,4\}$	$\{2,3,4\}$	4
$\{3,4\}$	$\{1,4\}$	$\{4\}$	5
$\{2,3,4\}$	$\{1,2,4\}$	$\{2,4\}$	5

Les états finaux sont ceux qui contiennent 4 (le seul état final de l'automate donné au départ).

Voici le diagramme de transitions correspondant :



Exercice 1.111, page 95. Énumérez les premières séquences des langages suivants.

1. $(\{ab\} \cup \{cd, \lambda\})^*$

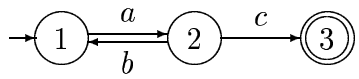
2. $(\{ab\} \circ \{cd, \lambda\})^*$
3. $\emptyset^*$

Solution. Pour les deux premiers problèmes, donnons les huit premières séquences (par ordre de longueur croissante). Pour le dernier, il est possible de donner toutes les séquences.

1. $(\{ab\} \cup \{cd, \lambda\})^*$
   $= \{ab, cd, \lambda\}^*$
   $= \{\lambda, ab, cd, abab, abcd, cdab, cdcd, ababab, \ldots\}$
2. $(\{ab\} \circ \{cd, \lambda\})^*$
   $= \{abcd, ab\}^*$
   $= \{\lambda, abcd, ab, abcdabcd, abcdab, ababcd, abab, abcdabcdabcd, \ldots\}$
3. $\emptyset^* = \{\lambda\}$

Exercice 1.114, page 97. Construisez l'automate qui accepte la fermeture du langage accepté par l'automate M suivant.

M :



Solution. En utilisant la méthode du théorème 1.112, construisons un automate M' tel que $L(M') = (L(M))^*$. Considérons que l'alphabet de M est $\{a, b, c\}$. Le quintuplet décrivant l'automate M est $(S, \{a, b, c\}, \rho, \iota, F)$, où

$S = \{1, 2, 3\}$,
$\rho = \{(1, a, 2), (2, b, 1), (2, c, 3)\}$,
$\iota = 1$,
$F = \{3\}$.

L'automate M' a la forme

$$M' = (S', \{a, b, c\}, \rho', \iota', F'),$$

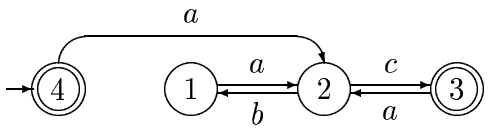
où

$\iota' = 4$ (un nouvel état),
$S' = S \cup \{\iota'\} = \{1, 2, 3, 4\}$,
$F' = F \cup \{\iota'\} = \{3, 4\}$

et

$$
\begin{aligned}
& \rho' \\
= \; & \rho \cup \{(p, x, q) : p \in F' \wedge (\iota, x, q) \in \rho\} \\
= \; & \rho \cup \{(p, x, q) : p \in \{3, 4\} \wedge (1, x, q) \in \{(1, a, 2), (2, b, 1), (2, c, 3)\}\ \} \\
= \; & \{(1, a, 2), (2, b, 1), (2, c, 3)\} \cup \{(3, a, 2), (4, a, 2)\} \\
= \; & \{(1, a, 2), (2, b, 1), (2, c, 3), (3, a, 2), (4, a, 2)\}
\end{aligned}
$$

Le diagramme de transitions de M' est le suivant :



Il est facile de voir que

$$L(M) = \{a(ba)^n c : n \in \mathbf{N}\} = \{(ab)^n ac : n \in \mathbf{N}\}.$$

Par conséquent, par le théorème 1.112,

$$L(M') = \{a(ba)^n c : n \in \mathbf{N}\}^* = \{(ab)^n ac : n \in \mathbf{N}\}^*.$$

Remarquez comment il serait difficile de décrire $L(M')$ sans utiliser l'étoile.

Exercice 1.116, page 98. Soit $\Sigma = \{a, b, c\}$. Parmi les termes suivants, identifiez ceux qui sont des expressions régulières sur Σ. Lorsqu'un terme n'est pas une expression régulière, dites pourquoi.

1. $(a \cup (b \circ c))$
2. $[a \cup (b \circ c)]$
3. b^*
4. $(b)^*$
5. $(((a \cup b)^* \circ b)$
6. $(a \cup b)^* \circ b$

Solution.

1. Oui. Il suffit d'appliquer la règle 2 de la définition 1.115 trois fois pour obtenir les trois expressions régulières a, b et c. En appliquant ensuite la règle 4 à b et c, on obtient $(b \circ c)$. Finalement, en appliquant la règle 3 à a et $(b \circ c)$, on obtient l'expression donnée.
2. Non. Les crochets carrés [ ] ne sont pas permis.

3. Oui. On l'obtient par une application de la règle 2, suivie d'une application de la règle 5.
4. Non. Pour que $(b)^*$ soit une expression régulière, il faudrait que (b) en soit une. Or, il n'y a aucune règle qui permet de générer (b) à partir de l'expression b.
5. Non. Les parenthèses sont mal équilibrées.
6. Non. Il manque les parenthèses externes. L'expression $((a \cup b)^* \circ b)$, elle, est une expression régulière.

Exercice 1.119, page 100. En utilisant la définition 1.117, dites quel est le langage représenté par les expressions régulières suivantes. Donnez les étapes menant à cette description du langage (comme il est fait à l'exemple 1.118). Simplifiez la description du langage autant que possible.

1. $(a \circ (a \cup b))$
2. $((a \circ a)^* \cup (b \cup b)^*)$

Solution. Les clauses mentionnées dans la solution sont celles de la définition 1.117.

1. $L((a \circ (a \cup b)))$
   $=$ $\langle$ Clause 4. $\rangle$
   $L(a) \circ L((a \cup b))$
   $=$ $\langle$ Clause 3. $\rangle$
   $L(a) \circ (L(a) \cup L(b))$
   $=$ $\langle$ Clause 2. $\rangle$
   $\{a\} \circ (\{a\} \cup \{b\})$
   $=$ $\langle$ Définition de l'union de langages. $\rangle$
   $\{a\} \circ \{a, b\}$
   $=$ $\langle$ Définition de la concaténation de langages. $\rangle$
   $\{aa, ab\}$.
2. $L(((a \circ a)^* \cup (b \cup b)^*))$
   $=$ $\langle$ Clause 3. $\rangle$
   $L((a \circ a)^*) \cup L((b \cup b)^*)$
   $=$ $\langle$ Clause 5. $\rangle$
   $(L((a \circ a)))^* \cup (L((b \cup b)))^*$
   $=$ $\langle$ Clauses 3 et 4. $\rangle$

$(L(a) \circ L(a))^* \cup (L(b) \cup L(b))^*$

$=$ $\langle$ Clause 2. $\rangle$

$(\{a\} \circ \{a\})^* \cup (\{b\} \cup \{b\})^*$

$=$ $\langle$ Définition de la concaténation et de l'union. $\rangle$

$\{aa\}^* \cup \{b\}^*$

Nous pourrions aussi continuer pour obtenir $\{a^{2n} : n \in \mathbf{N}\} \cup \{b^n : n \in \mathbf{N}\}$.

Exercice 1.120, page 100. Pour chacun des langages suivants, donnez une expression régulière qui le représente.

1. $\{a^j b a^k : j, k \in \mathbf{N}^+\}$
2. $\{w^n : n \in \mathbf{N} \wedge (w = ab \vee w = ba)\}$

Solution. Voici quelques réponses possibles. Toutes ces expressions représentent le bon langage, comme on peut le vérifier en appliquant la définition 1.117.

1. $((a \circ a^*) \circ b) \circ (a \circ a^*))$
   $(a \circ (a^* \circ (b \circ (a \circ a^*))))$
   $((((a \circ a^*) \circ b) \circ a) \circ a^*)$
2. $((a \circ b)^* \cup (b \circ a)^*)$
   $((b \circ a)^* \cup (a \circ b)^*)$

Théorème 1.122, page 100. (Partie 1)

Voici une preuve plus détaillée et meilleure que celle des notes. C'est une preuve par induction sur le nombre d'opérateurs de l'expression régulière considérée (les opérateurs sont les symboles $\cup, \circ, ^*$).

Il faut montrer que si un langage est représentable par une expression régulière, alors c'est un langage régulier. Puisqu'une expression régulière contient un nombre fini d'opérateurs (0 ou plus), ceci revient à montrer que pour tout $n \in \mathbf{N}$, toute expression régulière contenant au plus n opérateurs représente un langage régulier.

Soit $P(n)$ le prédicat

> toute expression régulière contenant au plus n opérateurs représente un langage régulier.

Il faut montrer qu'il est vrai pour tout $n \in \mathbf{N}$.

- Base d'induction : $n = 0$. Ce cas correspond aux expressions régulières qui n'ont aucun opérateur. Il y a deux sous-cas (selon la définition 1.115) :
  - L'expression régulière est $\emptyset$: Par la définition 1.117, $L(\emptyset) = \emptyset$, qui est un langage régulier, car il existe un automate fini qui le reconnaît.

- L'expression régulière est x, où $x \in \Sigma$: Par la définition 1.117, $L(x) = \{x\}$, qui est un langage régulier, car il existe un automate fini qui le reconnaît.

- Étape d'induction. Il faut montrer $P(k) \Rightarrow P(k+1)$, pour tout $k \in \mathbf{N}$. Supposons donc que les expressions régulières contenant au plus k opérateurs représentent un langage régulier (hypothèse d'induction). Il faut montrer que les expressions régulières contenant au plus $k+1$ opérateurs représentent un langage régulier. Il y a trois sous-cas :
  - L'expression régulière a la forme $(p \cup q)$ et contient $k+1$ opérateurs :

    $(p \cup q)$ contient $k+1$ opérateurs

    $\Rightarrow$ $\langle$ $\cup$ est un opérateur. $\rangle$

    p et q contiennent au plus k opérateurs chacunes

    $\Rightarrow$ $\langle$ Hypothèse d'induction. $\rangle$

    $L(p)$ et $L(q)$ sont des langages réguliers

    $\Rightarrow$ $\langle$ Théorème 1.99. $\rangle$

    $L(p) \cup L(q)$ est un langage régulier

    $\Rightarrow$ $\langle$ Par la définition 1.117, $L(p) \cup L(q) = L((p \cup q))$. $\rangle$

    $L((p \cup q))$ est un langage régulier
  - L'expression régulière a la forme $(p \circ q)$ et contient $k+1$ opérateurs : la preuve est similaire à celle qui précède.
  - L'expression régulière a la forme p^* et contient $k+1$ opérateurs : la preuve est similaire à celle des deux cas précédents. Donnons-la quand même.

    p^* contient $k+1$ opérateurs

    $\Rightarrow$ $\langle$ * est un opérateur. $\rangle$

    p contient k opérateurs

    $\Rightarrow$ $\langle$ Hypothèse d'induction. $\rangle$

    $L(p)$ est un langage régulier

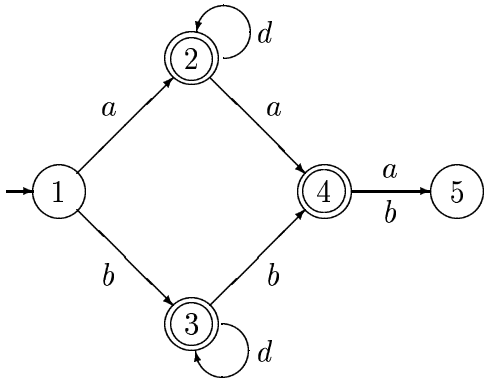
    $\Rightarrow$ $\langle$ Théorème 1.112. $\rangle$

    $(L(p))^*$ est un langage régulier

    $\Rightarrow$ $\langle$ Par la définition 1.117, $(L(p))^* = L(p^*)$. $\rangle$

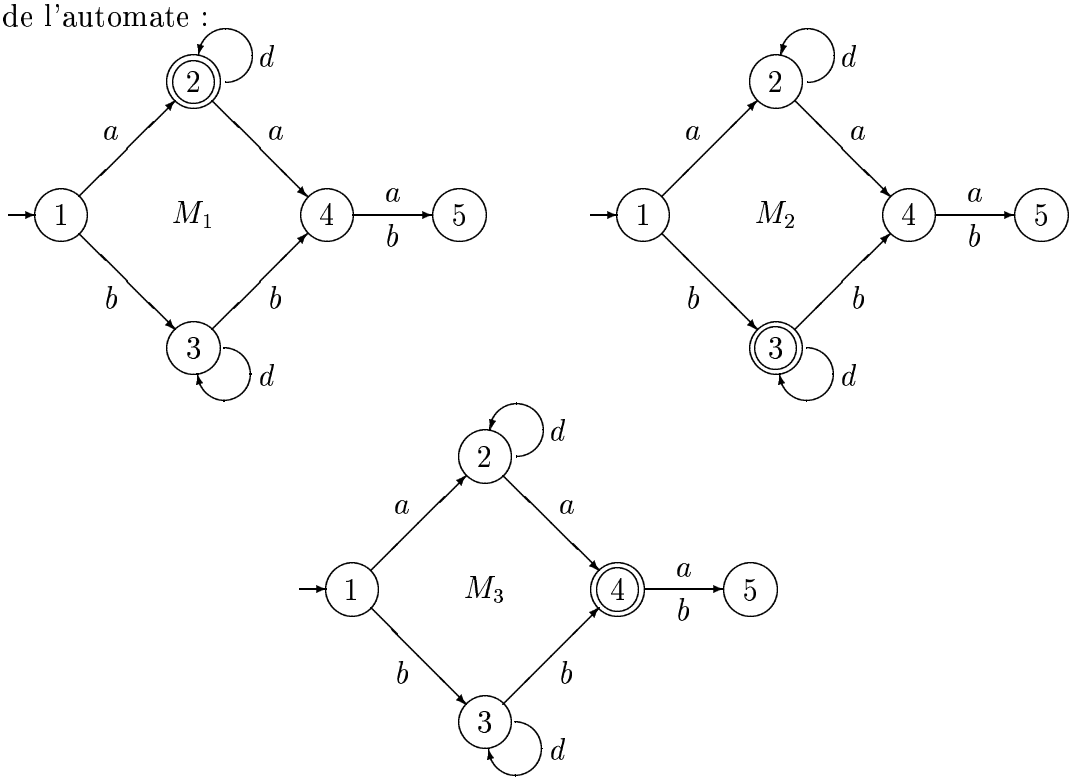
    $L(p^*)$ est un langage régulier

Exercice 1.124, page 106. Donnez une expression régulière qui représente le langage accepté par l'automate M suivant.

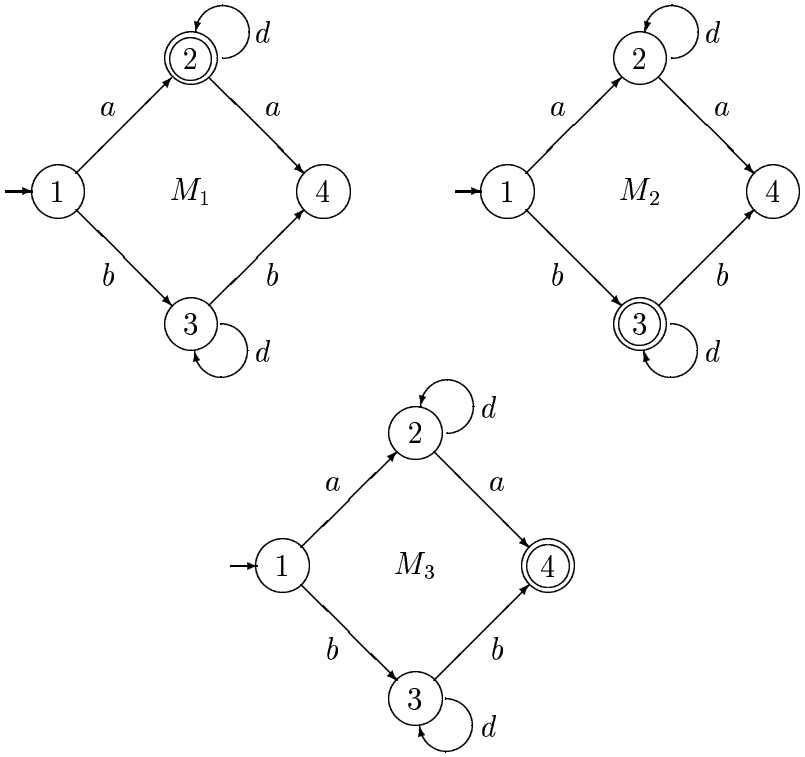


Solution. Procédons selon la méthode décrite dans la partie 2 du théorème 1.122. Nous remarquons qu'il serait possible de supprimer l'état 5 sans changer le langage accepté par l'automate ; ceci simplifierait la démarche subséquente. Nous appliquerons plutôt la méthode du théorème strictement, et la suppression se fera en temps et lieu.

- Cas 2 (du théorème) : Puisqu'il y a trois états finaux, il faut faire trois copies de l'automate :

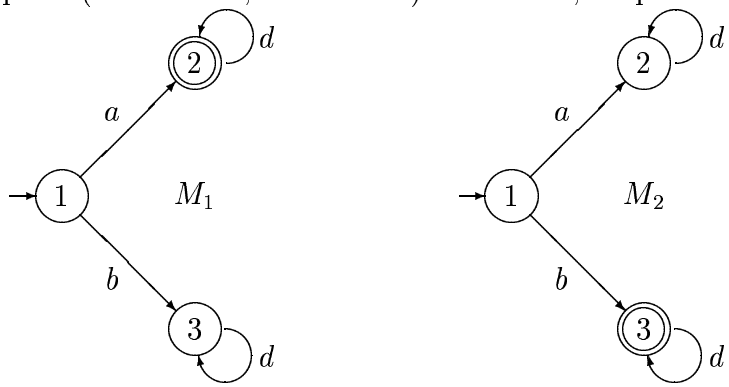


- Cas 3b(iii) : Les automates ont un seul état final. Il y a au plus un arc dans chaque direction pour chaque paire d'états. L'état 5 n'est pas initial, ni final ; comme il n'y a aucune paire (arc entrant, arc sortant) de l'état 5, on peut le supprimer.

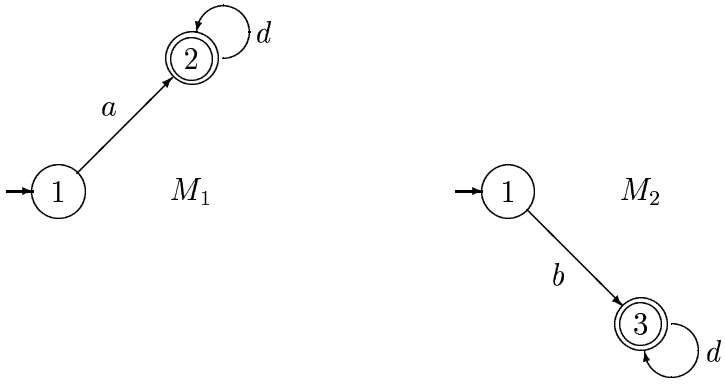


Traitons M_1 et M_2 en parallèle, car ils se ressemblent.

- Cas 3b(iii) : L'état 4 des deux automates n'est ni initial, ni final ; comme il n'y a aucune paire (arc entrant, arc sortant) de l'état 4, on peut le supprimer.



- Cas 3b(iii) : L'état 3 de M_1 et l'état 2 de M_2 ne sont pas initiaux, ni finaux ; comme il n'y a aucune paire (arc entrant, arc sortant) de ces états, on peut les supprimer.

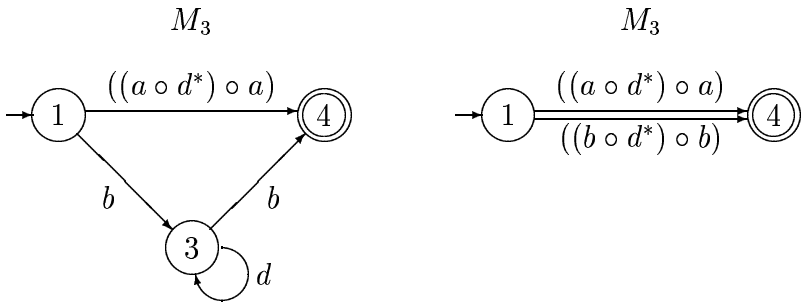


- Cas 3b(ii) : Il y a deux états, l'un initial, l'autre final.

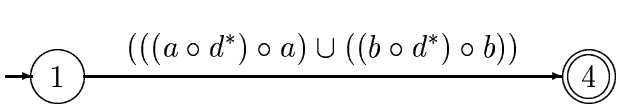
$$er_1 = (a \circ d^*) \qquad er_2 = (b \circ d^*)$$

Continuons le traitement de M_3.

- Cas 3b(iii) : L'état 2 de M_3 n'est pas initial, ni final ; il faut tenir compte de toutes les paires (arc entrant, arc sortant) de cet état avant de le supprimer (il n'y a qu'une telle paire). L'état 3 est ensuite supprimé de manière similaire.



- Cas 3a : Un seul état final, deux transitions de l'état 1 vers l'état 4.



- Cas 3b(ii) : $er_3 = (((a \circ d^*) \circ a) \cup ((b \circ d^*) \circ b))$

En regroupant les résultats, on a

$$\begin{aligned} & er \\ = \ & ((er_1 \cup er_2) \cup er_3) \\ = \ & (((a \circ d^*) \cup (b \circ d^*)) \cup (((a \circ d^*) \circ a) \cup ((b \circ d^*) \circ b))). \end{aligned}$$

Exercice 2.3, page 113. Que fait l'automate de l'exemple 2.2 en passant

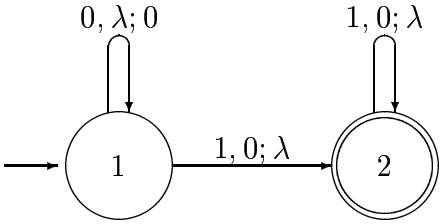
1. de l'état 2 à l'état 2 ?
2. de l'état 3 à l'état 3 ?

3. de l'état 3 à l'état 4 ?

Solution.

1. Lecture d'un a, empilement d'un a.
2. Lecture d'un b, dépilement d'un a.
3. Dépilement d'un #.

Exercice 2.6, page 115. Soit le diagramme de transitions d'un automate à pile :



1. Quelle est la définition formelle de cet automate (c'est-à-dire quels sont ses paramètres $S, \Sigma, \Gamma, T, \iota, F$) ?
2. Donnez la suite des configurations lorsque la séquence d'entrée est
   (a) 0011
   (b) 011
   (c) 001
3. Quel langage accepte-t-il ?

Solution.

1. $S = \{1, 2\}$ $\quad T = \{(1, 0, \lambda; 1, 0), (1, 1, 0; 2, \lambda), (2, 1, 0; 2, \lambda)\}$
   $\Sigma = \{0, 1\}$ $\quad \iota = 1$
   $\Gamma = \{0\}$ $\quad F = \{2\}$
   Les valeurs données pour Σ et Γ sont les valeurs minimales. Ces alphabets pourraient contenir d'autres symboles en plus de ceux qui sont donnés.
2. (a)

état	ruban	pile
1	$\underline{0}011$	
1	$0\underline{0}11$	0
1	$00\underline{1}1$	00
2	$001\underline{1}$	0
2	$0011\underline{\ }$	

   La séquence est acceptée, parce qu'elle est toute lue et que l'état atteint (2) est final.

(b)

état	ruban	pile
1	$\underline{0}11$	
1	$0\underline{1}1$	0
2	$01\underline{1}$	

La séquence est rejetée. En effet, l'entrée n'est pas toute lue et il n'y a plus de transition possible ; de plus, il n'y a pas d'autre exécution que celle-là.

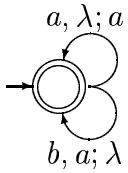
(c)

état	ruban	pile
1	$\underline{0}01$	
1	$0\underline{0}1$	0
1	$00\underline{1}$	00
2	$001\underline{\ }$	0

La séquence est acceptée (même si la pile n'est pas vide) car l'état atteint (2) est final et la séquence est toute lue.

3. Le langage accepté est $\{0^m1^n : m, n \in \mathbf{N}^+ \wedge m \geq n\}$. Il est facile de montrer que ce langage n'est pas régulier. Remarquons qu'on a bien $\mathbf{N}^+$ et non $\mathbf{N}$ car, pour passer de l'état 1 à l'état 2, il faut dépiler un 0 en lisant un 1, et pour qu'il y un 0 à dépiler, il faut en avoir lu un.

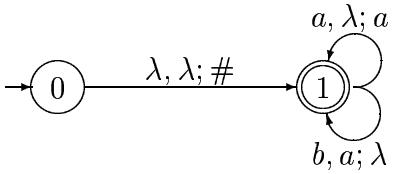
Exercice 2.9, page 118. Voici le diagramme de transitions d'un automate à pile.



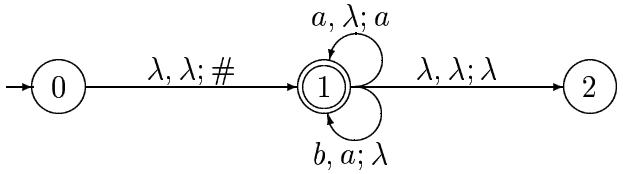
Transformez cet automate pour qu'il accepte le même langage en vidant toujours sa pile lorsqu'il accepte.

Solution. Nous considérons que Γ est le plus petit alphabet compatible avec le diagramme de transitions, ce qui donne $\Gamma = \{a\}$. Donnons à l'unique état l'étiquette 1, afin de pouvoir y référer dans les exemples d'exécution qui sont donnés ci-dessous. Appliquons l'algorithme décrit dans le théorème 2.8.

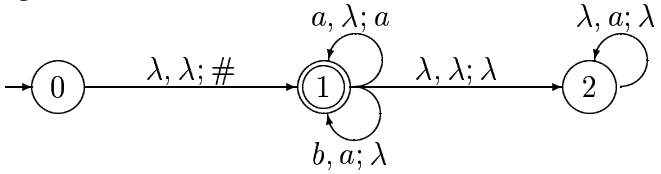
1. Ajouter un nouvel état initial ι' et une transition $(\iota', \lambda, \lambda; \iota, \#)$, où $\# \notin \Gamma$. Prenons $\iota' = 0$. Comme $\# \notin \Gamma$, nous pouvons garder ce symbole (sinon il faudrait en prendre un autre).



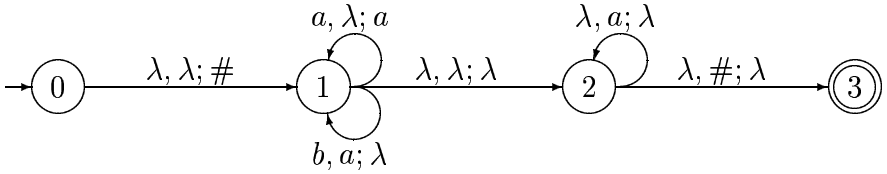
2. Ensuite, ajouter un nouvel état p. Puis, pour tous les $s \in F$, ajouter une transition $(s, \lambda, \lambda; p, \lambda)$. Prenons $p = 2$. Comme 1 est le seul état final, il n'y a qu'une transition à ajouter.



3. Pour tout $z \in \Gamma$, ajouter les transitions $(p, \lambda, z; p, \lambda)$. Comme Γ ne contient que le symbole a, il n'y a qu'une transition à ajouter. Si nous avions choisi un alphabet Γ plus grand, il y aurait d'autres transitions à ajouter.



4. Ajouter un nouvel état q qui devient le seul état final, ainsi qu'une transition $(p, \lambda, \#; q, \lambda)$. Prenons $q = 3$.



Nous avons choisi de remplacer ι', p et q par $0, 2$ et 3, respectivement, afin d'illustrer cette possibilité. Pour cet automate, nous aurions aussi pu utiliser les étiquettes ι', p et q, car elles ne sont pas employées dans l'automate donné.

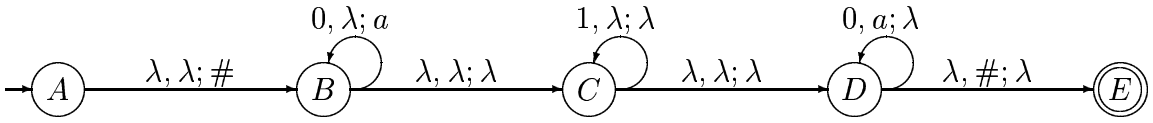
Voici comment la séquence aab est acceptée par l'automate donné (configurations de gauche) et par l'automate construit (configurations de droite).

état	ruban	pile
1	$\underline{a}ab$	
1	$a\underline{a}b$	a
1	$aa\underline{b}$	aa
1	$aab\_$	a

état	ruban	pile
0	$\underline{a}ab$	
1	$\underline{a}ab$	$\#$
1	$a\underline{a}b$	$\#a$
1	$aa\underline{b}$	$\#aa$
1	$aab\_$	$\#a$
2	$aab\_$	$\#a$
2	$aab\_$	$\#$
3	$aab\_$	

Exercice 2.11, page 119. Construisez un automate à pile acceptant le langage $\{0^n 1^m 0^n : m, n \in \mathbf{N}\}$.

Solution.



Nous avons choisi $S = \{A, B, C, D, E\}$ et $\Gamma = \{a, \#\}$, mais d'autres choix sont possibles, comme $S = \{0, 1, 2, 3, 4\}$ et $\Gamma = \{0, Z\}$. L'état A pourrait être final, afin d'accepter λ, mais λ est aussi acceptée par l'automate ci-dessus.

En plus de fournir la solution au problème, montrons que la séquence 0 est rejetée. Voici les deux seules exécutions possibles ; elles mènent toutes deux au rejet de 0. Dans le cas de l'exécution présentée à gauche, il n'est pas possible de passer à l'état final E, car il n'est pas possible de dépiler #. Dans le cas de l'exécution de droite, la séquence n'est pas toute lue. Comme *toutes les exécutions possibles* mènent au rejet de la séquence 0, 0 est rejetée.

état	ruban	pile
A	$\underline{0}$	
B	$\underline{0}$	#
B	$0\underline{\ }$	$\#a$
C	$0\underline{\ }$	$\#a$
D	$0\underline{\ }$	$\#a$

état	ruban	pile
A	$\underline{0}$	
B	$\underline{0}$	#
C	$\underline{0}$	#
D	$\underline{0}$	#
E	$\underline{0}$	

Il est facile de vérifier que les séquences 0011100 et 0000, par exemple, sont acceptées.

Exercice 2.15, page 121. Décrivez par une expression ensembliste le langage généré par la grammaire de l'exemple 2.14.

Solution. La première production appliquée doit être $S \to zMNz$. Le développement de M avec les productions $M \to z$ et $M \to aMa$ donne

$$M \Rightarrow z \qquad \text{et} \qquad M \Rightarrow aMa \Rightarrow \ldots \Rightarrow a^m z a^m.$$

On obtient donc des séquences de la forme $a^m z a^m$, avec $m \in \mathbf{N}$. De même, le développement de N avec les productions $N \to z$ et $N \to bNb$ donne

$$N \Rightarrow z \qquad \text{et} \qquad N \Rightarrow bNb \Rightarrow \ldots \Rightarrow b^n z b^n.$$

Ceci donne des séquences de la forme $b^n z b^n$, avec $n \in \mathbf{N}$. On a donc

$$L(G) = \{z a^m z a^m b^n z b^n z : m, n \in \mathbf{N}\}.$$

Exercice 2.20, page 122. Soit la grammaire $G = (\{S\}, \{+, -, *, /, (,), a\}, S, R)$ où R est l'ensemble des règles suivantes :

$$\begin{array}{lll} S \to S + S & S \to S * S & S \to (S) \\ S \to S - S & S \to S/S & S \to a \end{array}$$

Donnez une dérivation à gauche, une dérivation à droite et l'arbre de dérivation de la séquence $a * (a + a)$.

Solution.

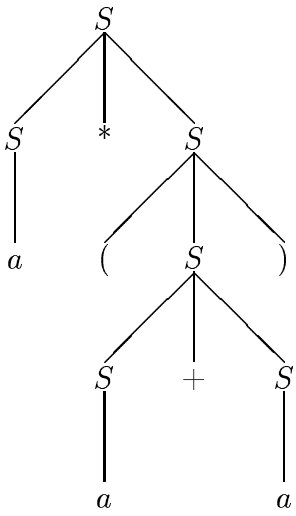
Dérivation à gauche

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow a * S \\ &\Rightarrow a * (S) \\ &\Rightarrow a * (S + S) \\ &\Rightarrow a * (a + S) \\ &\Rightarrow a * (a + a) \end{aligned}$$

Dérivation à droite

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow S * (S) \\ &\Rightarrow S * (S + S) \\ &\Rightarrow S * (S + a) \\ &\Rightarrow S * (a + a) \\ &\Rightarrow a * (a + a) \end{aligned}$$

Arbre de dérivation



Exercice 2.22, page 123.

Montrez que la grammaire G ci-dessous est ambiguë en produisant la séquence

si condition alors si condition alors instruction sinon instruction

par des dérivations dont les arbres sont différents.

$G = (\{S\}, \{\text{condition, si, alors, sinon, instruction}\}, S, R)$,

où R est l'ensemble des règles suivantes.

(1) $S \to$ si condition alors S
(2) $S \to$ si condition alors S sinon S
(3) $S \to$ instruction

Quel est le résultat du programme suivant, selon l'arbre de dérivation utilisé pour analyser l'instruction conditionnelle ?

$x := 1$; si $x > 5$ alors si $x < 10$ alors $x := x + 1$ sinon $x := x - 1$

Solution.

- Dérivation 1

  S

  $\Rightarrow$ $\langle$ Règle 1 $\rangle$

  si condition alors S

  $\Rightarrow$ $\langle$ Règle 2 $\rangle$

  si condition alors si condition alors S sinon S

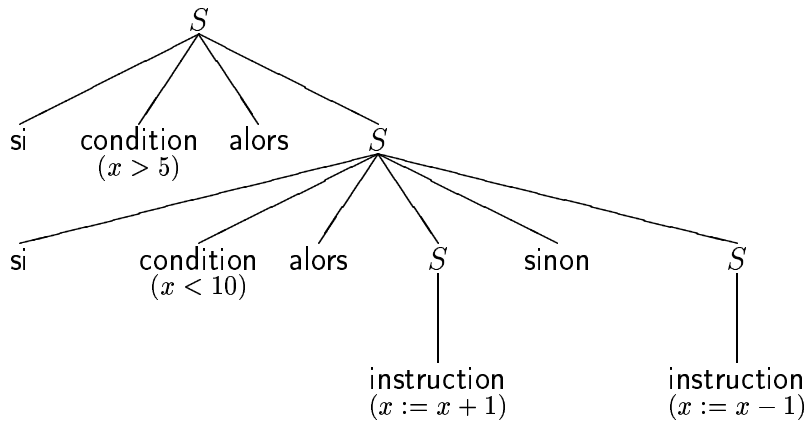
  $\Rightarrow$ $\langle$ Règle 3 $\rangle$

  si condition alors si condition alors instruction sinon S

  $\Rightarrow$ $\langle$ Règle 3 $\rangle$

  si condition alors si condition alors instruction sinon instruction

  Voici l'arbre de dérivation correspondant, auquel on a ajouté les conditions et les (sous-)instructions de l'instruction à évaluer.



  Le résultat de l'instruction à évaluer est $x = 1$.

- Dérivation 2

  S

  $\Rightarrow$ $\langle$ Règle 2 $\rangle$

  si condition alors S sinon S

  $\Rightarrow$ $\langle$ Règle 1 $\rangle$

  si condition alors si condition alors S sinon S

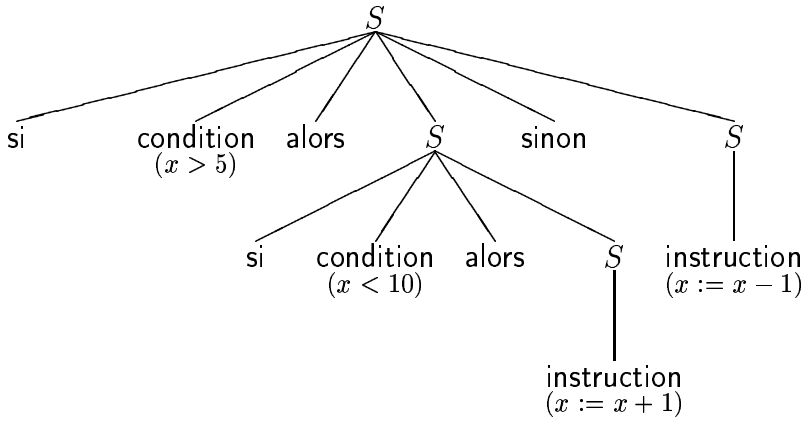
  $\Rightarrow$ $\langle$ Règle 3 $\rangle$

  si condition alors si condition alors instruction sinon S

  $\Rightarrow$ $\langle$ Règle 3 $\rangle$

si condition alors si condition alors instruction sinon instruction

Voici l'arbre de dérivation correspondant, auquel on a ajouté les conditions et les (sous-)instructions de l'instruction à évaluer.



Le résultat de l'instruction à évaluer est $x = 0$.

Afin de lever l'ambiguïté, certains langages se donnent des règles non contextuelles. Par exemple, en COBOL, le "sinon" est associé au "alors" le plus rapproché. Quelle que soit la règle, il est toujours possible de forcer le choix désiré au moyen d'une paire "début, fin". C'est ainsi que l'instruction

si $x > 5$ alors début si $x < 10$ alors $x := x + 1$ sinon $x := x - 1$ fin

permet d'obtenir le résultat correspondant au premier arbre de dérivation alors que l'instruction

si $x > 5$ alors début si $x < 10$ alors $x := x + 1$ fin sinon $x := x - 1$

mène au résultat correspondant au deuxième arbre.

Exercice 2.24, page 124. Construisez une grammaire non contextuelle qui génère le langage $\{a^m b^n c^m : m \in \mathbf{N} \wedge n \in \mathbf{N}^+\}$.

Solution.

$$S \to aSc \qquad S \to B \qquad B \to bB \qquad B \to b$$

Les règles $S \to aSc$ et $S \to B$ permettent de générer les séquences de la forme $a^m B c^m$, où $m \in \mathbf{N}$. Les deux autres règles génèrent les séquences de la forme b^n, où $n \in \mathbf{N}^+$.

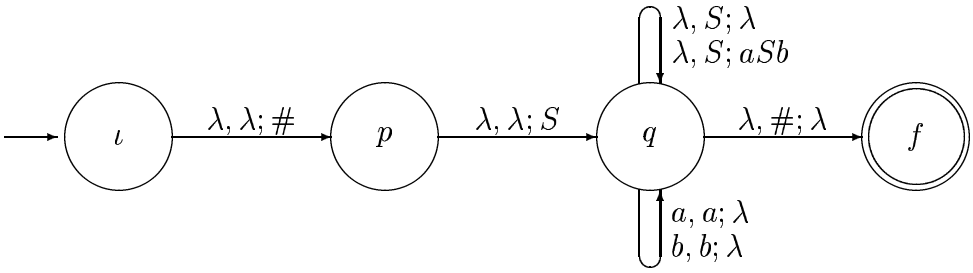
Exercice 2.28, page 128. Trouvez une grammaire qui génère le langage $\{a^n b^n : n \in \mathbf{N}\}$, puis utilisez la technique du théorème 2.26 pour construire un automate à

pile qui accepte ce langage. Donnez la suite de configurations menant à l'acceptation de la séquence ab.

Solution. La grammaire est

$$S \to aSb \qquad S \to \lambda$$

L'automate obtenu en appliquant le théorème 2.26 est le suivant :



Voici finalement la suite de configurations demandée :

état	ruban	pile
ι	$\underline{a}b$	
p	$\underline{a}b$	$\#$
q	$\underline{a}b$	$S\#$
q	$\underline{a}b$	$aSb\#$
q	$a\underline{b}$	$Sb\#$
q	$a\underline{b}$	$b\#$
q	$ab\_$	$\#$
f	$ab\_$	

Comparez cet automate avec celui de l'exemple 2.2, qui accepte le même langage.

Exercice 2.34, page 132. Soit la grammaire G suivante :

$$S \to aSb \qquad N \to S$$
$$S \to dNa \qquad N \to \lambda$$

1. Quel est le langage généré par cette grammaire ?
2. Cette grammaire ne génère pas λ, bien qu'elle contienne une règle-λ. Utilisez l'algorithme présenté au théorème 2.32 pour trouver une grammaire équivalente sans règle-λ.

Solution.

1. Il est plus facile de voir quel est le langage généré par la grammaire si on la simplifie d'abord. On remarque qu'il est possible de se débarrasser des deux règles $N \to S$ et $N \to \lambda$ en substituant N, dans la règle $S \to dNa$, par le côté droit de ces deux règles. La résultat est la grammaire

   $$S \to aSb \qquad S \to dSa \qquad S \to da$$

   On voit alors que toute séquence générée se divise en deux parties qui sont presque des images miroir, sauf qu'un a dans la première partie correspond à un b dans la deuxième (à cause de la règle $S \to aSb$) et qu'un d dans la première partie correspond à un a dans la deuxième (à cause des deux autres règles) ; la première partie se termine par un d (à cause de la règle $S \to da$, seule règle qui permet de terminer). Par exemple, pour la séquence *aadadababb*, générée par cette grammaire, les deux parties sont *aadad* et *ababb*. Le langage généré par la grammaire est

   > $\{vdav' : v \in \{a,d\}^*$ et v' est la séquence v écrite à l'envers en y remplaçant les a par des b et les d par des $a\}$.

2. En effet, la grammaire donnée ne génère pas λ, puisque toute séquence débute soit par un a (à cause de la règle $S \to aSb$), soit par un d (à cause de la règle $S \to dNa$). C'est encore plus évident quand on considère la grammaire de l'item 1, car elle ne contient pas de règle-λ. Appliquons le théorème.

   (a) Il y a une seule chaîne-λ, soit $N \to \lambda$, et donc $U = \{N\}$.

   (b) Seule la production $S \to dNa$ a un symbole de U à droite. En y supprimant le symbole N, on obtient la règle $S \to da$, qu'il faut ajouter à la grammaire.

   (c) On enlève les règles-λ. Le résultat final est la grammaire

   $$S \to aSb \qquad S \to dNa \qquad S \to da \qquad N \to S$$

   Cette grammaire n'est pas aussi simple que celle que nous avons trouvée en répondant à la première partie de la question. L'intérêt du théorème 2.32, c'est qu'il fournit un *algorithme* (pour lequel on pourrait donc écrire un programme) qui permet de faire la transformation automatiquement.

Exercice 2.37, page 135. Transformez la grammaire obtenue à l'exercice 2.34 en grammaire sous forme normale de Chomsky.

Solution. La grammaire obtenue à l'exercice 2.34 est

$$S \to aSb \qquad S \to dNa \qquad S \to da \qquad N \to S$$

Appliquons le théorème 2.36.

1. Remplacer chaque symbole terminal x par un nouveau symbole non terminal X et ajouter la règle $X \to x$.

   $S \to ASB \quad S \to DA \quad A \to a \quad D \to d$
   $S \to DNA \quad N \to S \quad B \to b$

2. Transformer le côté droit des règles ayant plus de 2 symboles non terminaux.

   $S \to AR \quad R \to SB \quad S \to DA \quad A \to a \quad D \to d$
   $S \to DP \quad P \to NA \quad N \to S \quad B \to b$

3. Éliminer les règles ayant un seul symbole non terminal du côté droit. Il y a une seule chaîne impliquant de telles règles, soit $N \to S$. À cause des règles $S \to AR$, $S \to DP$ et $S \to DA$, il faut ajouter les règles $N \to AR$, $N \to DP$ et $N \to DA$.

   $S \to AR \quad N \to AR \quad R \to SB \quad A \to a$
   $S \to DP \quad N \to DP \quad P \to NA \quad B \to b$
   $S \to DA \quad N \to DA \quad D \to d$

Exercice 2.44, page 145. Montrez que le langage $L = \{a^j b^k a^j b^k a^j b^k : j, k \in \mathbf{N}^+\}$ n'est pas un langage non contextuel.

Solution. Le langage L est infini. Nous pouvons appliquer le corollaire 2.42. D'après le corollaire, il suffit de montrer que pour toute séquence $svuwt$, où $v \neq \lambda$ ou $w \neq \lambda$, il existe un $n \in \mathbf{N}^+$ tel que $sv^n uw^n t \notin L$.

Soit une séquence quelconque $a^j b^k a^j b^k a^j b^k \in L$. Supposons qu'elle est décomposée sous la forme $svuwt$, où $v \neq \lambda$ ou $w \neq \lambda$. Comme nous ne connaissons pas s, t, u, v, w, il faut étudier les divers cas qui peuvent se présenter.

1. Notons qu'une séquence de L contient exactement 3 sous-séquences de a séparées par des b (au moins un b, car $k \in \mathbf{N}^+$). Si la sous-séquence v contient des a et des b, alors v^4 contient 4 suites de a séparées par des b. Donc $sv^4uw^4t \notin L$, car cette séquence contient plus de 3 sous-séquences de a séparées par des b.
2. De la même manière, si w contient à la fois des a et des b, alors $sv^4uw^4t \notin L$.
3. Supposons $v = a^i$ pour un certain $i > 0$ (c'est-à-dire que v contient seulement des a). La situation est l'une des trois suivantes :
   - $a\underbrace{\ldots}_{v}ab\ldots ba\ldots ab\ldots ba\ldots ab\ldots b$
   - $a\ldots ab\ldots ba\underbrace{\ldots}_{v}ab\ldots ba\ldots ab\ldots b$
   - $a\ldots ab\ldots ba\ldots ab\ldots ba\underbrace{\ldots}_{v}ab\ldots b$

   c'est-à-dire que v contient des a venant d'une seule des trois séries de a. Considérons sv^2uw^2t. Pour que cette séquence soit une séquence de L, il faudrait

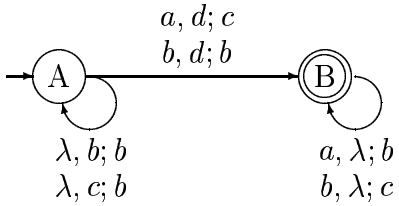
que w contienne des a venant des deux autres séries de a, sans contenir de b (à cause de la partie 2), ce qui est impossible, puisqu'il y a au moins un b entre deux séries de a.

4. De même, si $v = b^i, i > 0$, on a $sv^2uw^2t \notin L$.
5. En procédant comme en 3 et 4, on voit que w ne peut contenir seulement des a ou seulement des b.

Comme $v \neq \lambda$ ou $w \neq \lambda$, nous avons considéré tous les cas possibles.

Exercice 2.47, page 149. Pour chacun des automates à pile qui suivent, dites s'il est déterministe et expliquez pourquoi. Dans le cas où un automate n'est pas déterministe, donnez toutes les situations où le non déterminisme se manifeste.

1. $M_1 = (\{A, B\}, \{a, b\}, \{b, c, d\}, T_1, A, \{B\})$ où l'ensemble de transitions T_1 est décrit par le diagramme de transitions qui suit.



Solution. L'automate M_1 est déterministe, car il y a une et une seule transition possible pour chaque triplet (état, symbole d'entrée, symbole de pile) ; cette transition est donnée dans la table ci-dessous.

État	Entrée	Pile	Transition
A	a	b	$(A, \lambda, b; A, b)$
A	a	c	$(A, \lambda, c; A, b)$
A	a	d	$(A, a, d; B, c)$
A	b	b	$(A, \lambda, b; A, b)$
A	b	c	$(A, \lambda, c; A, b)$
A	b	d	$(A, b, d; B, b)$
B	a	b	$(B, a, \lambda; B, b)$
B	a	c	$(B, a, \lambda; B, b)$
B	a	d	$(B, a, \lambda; B, b)$
B	b	b	$(B, b, \lambda; B, c)$
B	b	c	$(B, b, \lambda; B, c)$
B	b	d	$(B, b, \lambda; B, c)$

2. $M_2 = (\{A, B, C\}, \{a, b\}, \{a, b\}, T_2, A, \{A, B\})$ avec

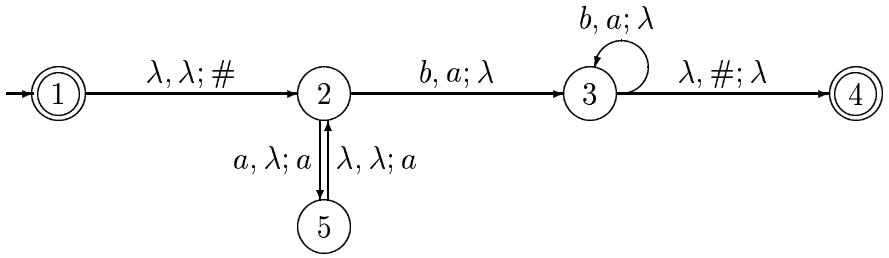
$$T_2 = \{(A, b, \lambda; A, b), (A, a, b; B, a), (B, a, \lambda; A, a), \\ (B, \lambda, a; C, \lambda), (C, a, b; C, a), (C, \lambda, \lambda; B, \lambda)\}.$$

Solution. L'automate M_2 n'est pas déterministe, car il n'est pas totalement défini (par exemple, il n'y a pas de transition possible lorsque l'état est A, le symbole d'entrée a et le symbole de pile a). De plus, il est ambigu (par exemple, les deux transitions $(B, a, \lambda; A, a)$, $(B, \lambda, a; C, \lambda)$ sont possibles lorsque l'état est B, le symbole d'entrée a et le symbole de pile a). Notez qu'une seule des deux raisons suffit.

La table suivante donne les transitions possibles pour chaque configuration. Un simple coup d'oeil à la table permet de trouver toutes les situations où le non-déterminisme se manifeste (deux cas de partialité et deux cas d'ambigüité).

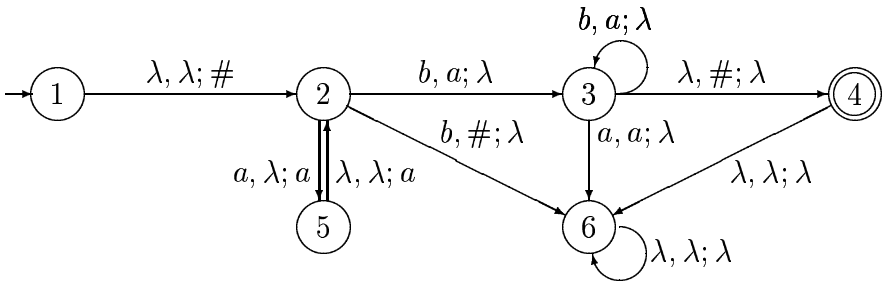
État	Entrée	Pile	Transition
A	a	a	
A	a	b	$(A, a, b; B, a)$
A	b	a	$(A, b, \lambda; A, b)$
A	b	b	$(A, b, \lambda; A, b)$
B	a	a	$(B, a, \lambda; A, a), (B, \lambda, a; C, \lambda)$
B	a	b	$(B, a, \lambda; A, a)$
B	b	a	$(B, \lambda, a; C, \lambda)$
B	b	b	
C	a	a	$(C, \lambda, \lambda; B, \lambda)$
C	a	b	$(C, \lambda, \lambda; B, \lambda), (C, a, b; C, a)$
C	b	a	$(C, \lambda, \lambda; B, \lambda)$
C	b	b	$(C, \lambda, \lambda; B, \lambda)$

Exercice 2.48, page 150. Soit M un automate à pile dont l'alphabet de ruban est $\{a, b\}$, l'alphabet de pile $\{a, \#\}$ et le diagramme de transitions



Donnez un automate déterministe qui accepte le même langage. Dites quel est le langage accepté par cet automate.

Solution. On note que l'automate n'est pas ambigu. Il suffit donc d'ajouter un état puits pour recevoir les cas non traités.



Le langage accepté par M est

$$L(M) = \{a^m b^{2m} : m \in \mathbf{N}\}.$$

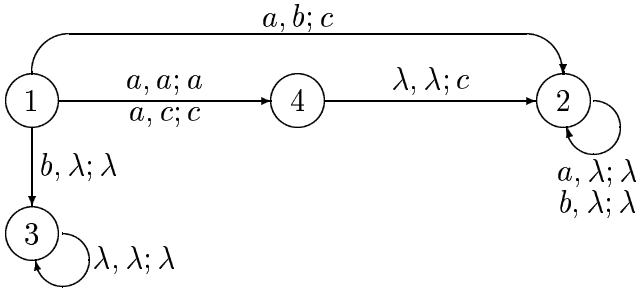
En effet, M marque d'abord le fond de la pile par #. Ensuite, à chaque passage dans la boucle 2–5, il lit un a et en empile deux. Pour revoir le # au fond de la pile et passer dans l'état final 4, il doit lire un b pour chaque a empilé. Il doit donc lire deux b pour chaque a lu. Pour passer de l'état 1 à l'état 4, il doit lire au moins un a, puisqu'il doit en dépiler un pour passer de 2 à 3. Les séquences qui mènent de 1 à 4 ont donc la forme $a^m b^{2m}$, où $m \in \mathbf{N}^+$. Comme l'état initial est aussi final, l'automate accepte λ, d'où le langage donné ci-dessus.

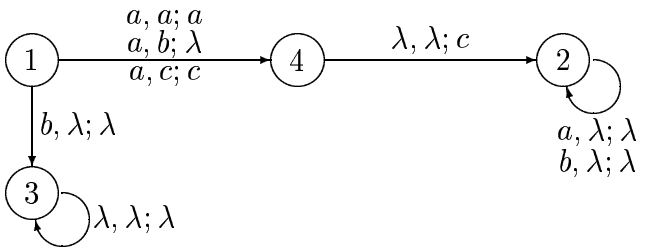
Exercice 2.49, page 150. Dessinez la partie appropriée d'un diagramme de transitions d'un automate à pile déterministe qui passe de l'état 1 à l'état 2 (en une ou plusieurs étapes) tout en lisant un a, en empilant un c et

$$\begin{cases} \text{en dépilant un } b \text{ si } b \text{ est sur la pile,} \\ \text{en ne dépilant rien dans le cas contraire.} \end{cases}$$

Si le symbole d'entrée n'est pas un a, l'automate passe de l'état 1 à l'état poubelle 3 en lisant le b. Dans l'état 2, l'automate finit de lire la séquence d'entrée, sans toucher à la pile. Supposez que $\Sigma = \{a, b\}$, $\Gamma = \{a, b, c\}$ et que la pile n'est pas vide dans l'état 1.

Solution. Voici deux solutions possibles. Notez qu'il ne serait pas correct de remplacer les transitions $(1, a, a; 4, a)$ et $(1, a, c; 4, c)$ par $(1, a, \lambda; 4, \lambda)$, car l'automate serait alors ambigu.



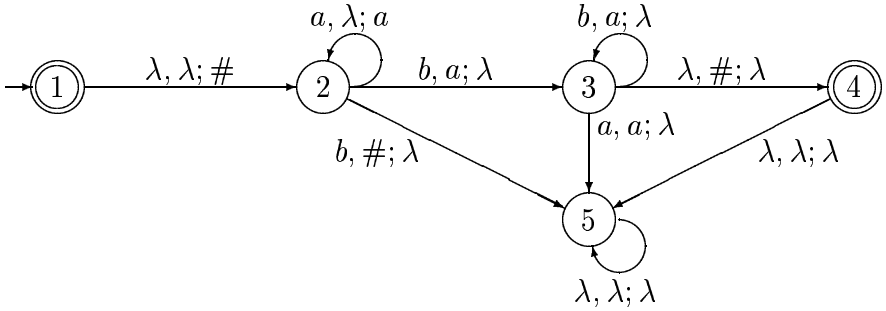


Exercice 2.51, page 155.

1. Donnez un exemple d'un langage non contextuel déterministe.
2. Donnez un exemple d'un langage non contextuel qui n'est pas un langage non contextuel déterministe.

Solution.

1. Le langage $\{a^n b^n : n \in \mathbf{N}\}$ est non contextuel déterministe. En effet, ce langage est accepté par l'automate à pile déterministe suivant (pour lequel $\Sigma = \{a, b\}$ et $\Gamma = \{a, \#\}$) :



   Notez que cet automate a été obtenu en rendant déterministe l'automate de l'exemple 2.2.

2. Le langage $\{a^n b^n : n \in \mathbf{N}^+\} \cup \{a^n b^{2n} : n \in \mathbf{N}^+\}$ est non contextuel mais n'est accepté par aucun automate à pile déterministe (c'est le langage utilisé dans la preuve du théorème 2.50).

Exercice 2.54, page 160. Voici une grammaire $G = (\{S\}, \{a, b, c\}, S, R)$ où R est l'ensemble des règles qui suivent.

$$S \to aSc \qquad S \to bSc \qquad S \to \lambda$$

1. Dites quel est $L(G)$.

2. Construisez la table $LL(1)$ de cette grammaire.
3. Donnez les résultats imprimés par la procédure LL(1) de la page 158 lorsqu'elle analyse les séquences suivantes au moyen de la table construite en 2. Dites si les séquences sont acceptées ou rejetées.
   (a) abcc
   (b) abc

Solution.

1. $L(G) = \{wc^n : w \in \{a,b\}^* \wedge |w| = n\}$. En effet, toute séquence du langage peut être divisée en deux moitiés de longueurs égales. La première moitié est une séquence arbitraire de a et de b alors que la deuxième moitié ne contient que des c.
2. Voici la table demandée. Le seul cas qui peut sembler étrange à première vue, c'est celui de la case (S, c) ; on pourrait croire qu'il s'agit d'un cas d'erreur. Cependant, on s'aperçoit vite (surtout après avoir examiné les exemples de l'item 3) que c'est bien λ qu'il faut mettre dans cette case ; cela a pour effet de faire disparaître le symbole S sur la pile, afin de permettre à la procédure LL(1) d'aller voir s'il y a un c sous ce S.

	a	b	c	FIN
S	aSc	bSc	λ	λ

3. Les résultats pour la séquence $abcc$ sont ceux de gauche. La séquence est acceptée ; on voit facilement qu'elle est générée par la grammaire. La table de droite contient les résultats pour la séquence abc qui, elle, est rejetée.

	Pile	Symbole	À lire
1	S	a	bccFIN
2	aSc	a	bccFIN
3	Sc	b	ccFIN
4	$bScc$	b	ccFIN
5	Scc	c	cFIN
6	cc	c	cFIN
7	c	c	FIN
8		FIN	

	Pile	Symbole	À lire
1	S	a	bcFIN
2	aSc	a	bcFIN
3	Sc	b	cFIN
4	$bScc$	b	cFIN
5	Scc	c	FIN
6	cc	c	FIN
7	c	FIN	

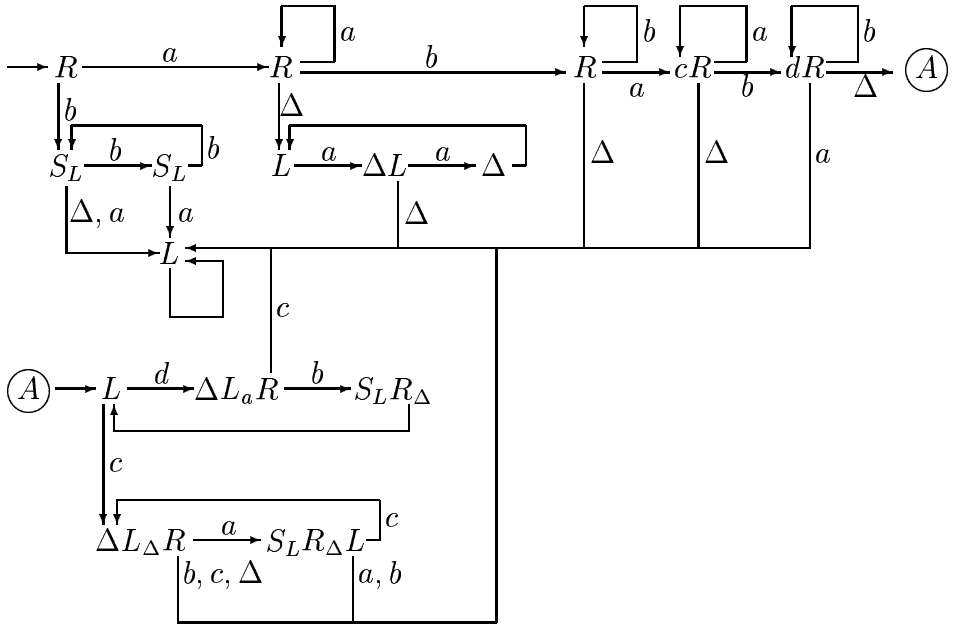
Exercice 3.18, page 190. Construisez une machine de Turing qui accepte

$\{a^m b^n a^m b^n : m, n \in \mathbf{N}\}$.

Solution. Voici une solution différente de celle qui a été donnée en classe. Bien que cette dernière soit plus simple, la solution ci-dessous est quand même intéressante et illustre une autre technique.

Comme la solution donnée en classe, la présente solution traite séparément les cas où il n'y a que des a, les cas où il n'y a que des b et les cas où il y a des a et des b.

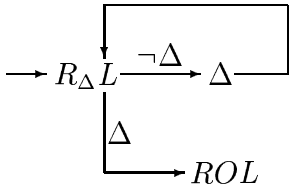
- Les cas où il n'y a que des a ou que des b sont traités de la même manière dans les deux solutions, excepté que l'une d'elles utilise une boucle de rejet avec un L et l'autre avec un R, ce qui est essentiellement la même chose (un a, un b ou un Δ pourrait aussi être utilisé).
- Lorsqu'il y a des a et des b, il n'est pas possible d'éliminer les a en comparant ceux qui viennent des deux séries de a, car cela peut conduire à l'acceptation d'une séquence qui doit être rejetée. Par exemple, l'élimination des deux séries de a (de longueurs égales) dans la séquence $aabbbaab$ conduit à la séquence $bbbb$. Or, celle-ci ne contient plus l'information que les deux séries de b n'avaient pas la même longueur initialement et que la séquence doit être rejetée.
  Pour éviter cela, la machine ci-dessous remplace les a de la deuxième série par des c et les b de la deuxième série par des d avant de comparer les a avec les c et les b avec les d.



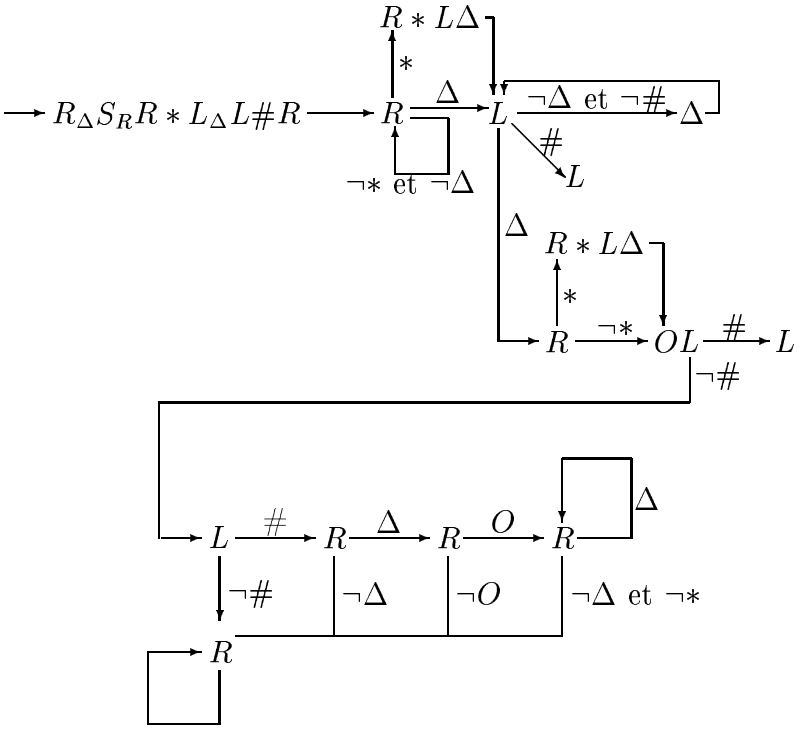
Exercice 3.23, page 196.

Soit $\Sigma = \{a, b\}$ l'alphabet d'entrée et $\Gamma = \{a, b, \Delta, O\}$ l'alphabet de ruban de la machine M suivante. M accepte $\{a, b\}^*$ avec message. Utilisez la deuxième partie du

théorème 3.20 pour construire une machine de Turing M' qui accepte par arrêt et telle que $L(M') = L(M)$.



Solution. La chose importante à remarquer, c'est qu'il faut faire l'expansion de R_Δ (regardez à la page 183) avant d'appliquer le théorème 3.20. C'est que l'instruction R_Δ amène la tête de lecture/écriture à passer par dessus le symbole $*$ qui marque la fin de la séquence sur le ruban, alors qu'il faut traiter de manière spéciale les cas où la tête se retrouve sur ce symbole.



Chapitre 5

Solution de quelques problèmes

5.1 Chapitre 0

Problème 1, page 11. Vrai ou faux ?

(a) Un ensemble peut contenir des éléments qui sont non corrélés. Par exemple $\{a, 1, \text{Fred}, \text{New Jersey}\}$. Vrai.

(b) Si $A \subseteq B$ et $B - A \neq \emptyset$ alors $A \subset B$. Vrai.

(c) $A = B$ si $\wp(A) = \wp(B)$. Vrai.

(d) Le domaine d'une relation est toujours égal à son ensemble de départ. Faux.

Problème 2, page 11. Dites si les expressions suivantes sont vraies ou fausses. Expliquez.

(a) $\{\{1, 2\}\} = \{1, 2\}$
Faux. L'ensemble $\{\{1, 2\}\}$ contient un seul élément (soit $\{1, 2\}$), alors que $\{1, 2\}$ en contient deux (1 et 2).

(b) $\{1, 2\} \subseteq \{\{1, 2\}\}$
Faux. Les éléments de $\{1, 2\}$ (c'est-à-dire 1 et 2) ne sont pas éléments de $\{\{1, 2\}\}$, dont le seul élément est $\{1, 2\}$.

(c) $\{1, 2\} \in \{\{1, 2\}\}$
Vrai. L'ensemble $\{1, 2\}$ est un élément de l'ensemble $\{\{1, 2\}\}$; c'est d'ailleurs son seul élément.

(d) $\{1, 2\} \in \mathbf{N}$
Faux. L'ensemble $\{1, 2\}$ n'est pas un élément de l'ensemble $\mathbf{N}$, car ce dernier ne contient que des nombres.

(e) $\emptyset \notin \{1, 2\}$
Vrai. L'ensemble vide n'est pas un élément de l'ensemble $\{1, 2\}$, qui a comme seuls éléments 1 et 2.

(f) $\{1, 2\} \subseteq \{a, b, 1, 2, 3\}$
Vrai. Tous les éléments de $\{1, 2\}$ sont des éléments de $\{a, b, 1, 2, 3\}$.

(g) $\{1, 2\} \not\subseteq \{1, a, b\}$
Vrai. L'élément 2 de $\{1, 2\}$ n'est pas un élément de $\{1, a, b\}$.

(h) $\{n \in \mathbf{N} : \exists m \in \mathbf{N} : n = m^2\} = \{n \in \mathbf{N} : \exists m \in \mathbf{N} : m = n^2\}$
Faux. L'ensemble de gauche est l'ensemble des carrés. L'ensemble de droite est l'ensemble des nombres naturels $\mathbf{N}$.

(i) $\{n \in \mathbf{N} : \exists m \in \mathbf{N} : n = 1 + m \text{ modulo } 2\} \subseteq \{1, 2\}$
Vrai. L'opérateur modulo est un opérateur multiplicatif et il a précédence sur l'addition. L'expression $1 + m$ modulo 2 se lit donc comme $1 + (m \text{ modulo } 2)$. Cette expression ne peut prendre que deux valeurs, soit 1 et 2.

Problème 3, page 12. Dites si chacun des ensembles suivants est un ensemble puissance d'un ensemble ? Si oui, quel est-il ?

(a) $\emptyset$ Non.

(b) $\{\emptyset, \{a\}\}$ Oui, $\{a\}$.

(c) $\{\emptyset, \{a\}, \{\emptyset, a\}\}$ Non.

(d) $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ Oui, $\{a, b\}$.

L'exercice 0.57 permet de répondre facilement pour le premier et le troisième cas. En effet, cet exercice montre que le nombre d'éléments de l'ensemble puissance d'un ensemble fini S est une puissance de 2.

Problème 4, page 12. Soit $A \subseteq U$. Complétez le tableau qui suit résumant les propriétés des opérations sur les ensembles :

Propriété	Nom
$A \cup \emptyset = A$ $A \cap U = A$	Élément neutre
$A \cup U = U$ $A \cap \emptyset = \emptyset$	Élément absorbant
$A \cup A = A$ $A \cap A = A$	Idempotence
$\overline{(\overline{A})} = A$	Involution
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutativité
$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	Associativité
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributivité
$\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$	Lois de De Morgan

Problème 5, page 13. Soit $A = \{1, 2, 3\}$. Calculez $A \cap \wp(A)$.

Solution. $A \cap \wp(A) = \{1, 2, 3\} \cap \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\} = \emptyset$.

Problème 6, page 13. Quel est $\mathbf{N} \cap \wp(\mathbf{N})$?

Solution. $\mathbf{N} \cap \wp(\mathbf{N}) = \emptyset$. En effet, les éléments de $\mathbf{N}$ sont des entiers naturels, alors que ceux de $\wp(\mathbf{N})$ sont des ensembles d'entiers naturels. L'ensemble $\mathbf{N}$ et l'ensemble $\wp(\mathbf{N})$ n'ont donc aucun élément en commun.

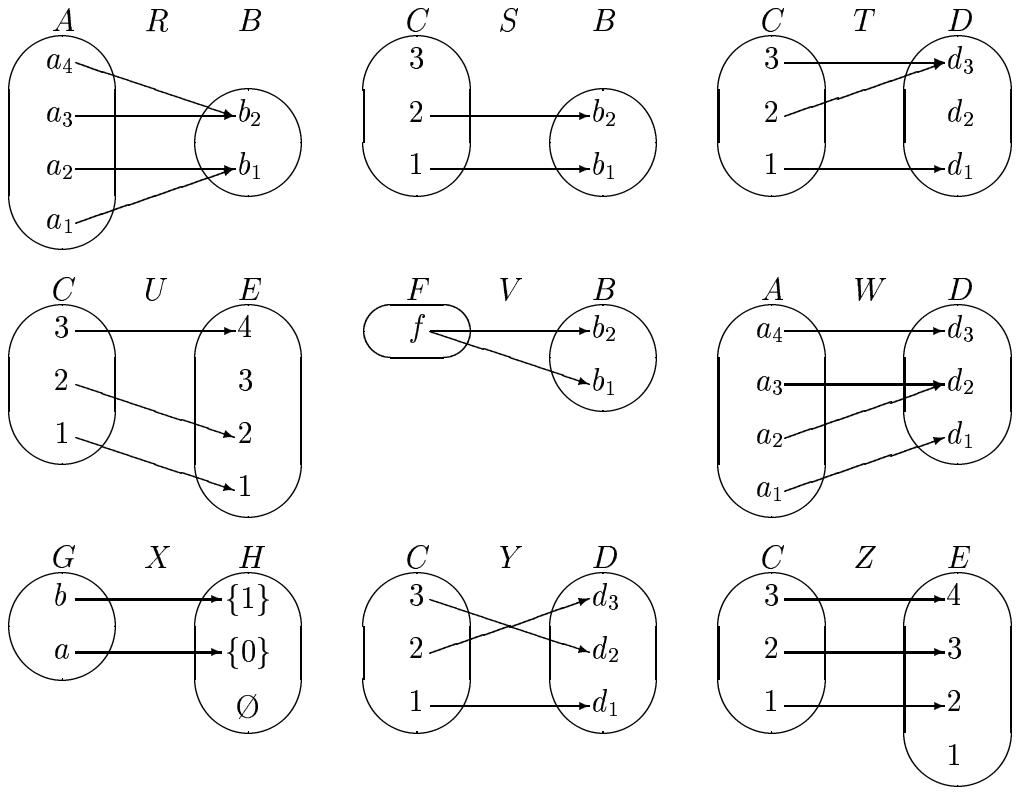
Problème 7, page 13. Calculez les expressions suivantes.

- $\{\emptyset, a\} \times \{1, 2\} \times \{i, j\}$
- $\{a, b\}^3$

Solution.

- $\{\emptyset, a\} \times \{1, 2\} \times \{i, j\} = \{(\emptyset, 1, i),\ (\emptyset, 1, j),\ (\emptyset, 2, i),\ (\emptyset, 2, j),$
  $(a, 1, i),\ (a, 1, j),\ (a, 2, i),\ (a, 2, j)\}$
- $\{a, b\}^3 = \{a, b\} \times \{a, b\} \times \{a, b\} = \{(a, a, a),\ (a, a, b),\ (a, b, a),\ (a, b, b),$
  $(b, a, a),\ (b, a, b),\ (b, b, a),\ (b, b, b)\}$

Problème 8, page 13. Dites si chacune des relations suivantes est une fonction. Dans l'affirmative, dites aussi les propriétés que la fonction possède (injectivité, surjectivité, bijectivité).



Solution. R est une fonction surjective,
S n'est pas une fonction, car elle n'est pas totale (3 n'est pas dans son domaine),
T est une fonction,
U est une fonction injective,
V n'est pas une fonction (f est à l'origine de deux arcs),
W est une fonction surjective,
X est une fonction injective,
Y est une fonction bijective,
Z est une fonction injective.

Problème 1, page 21. Transformez en implications les propositions suivantes.

(a) Si l'atmosphère était composée de 100% d'oxygène alors elle serait irrespirable pour beaucoup d'organismes.

(b) Si $A \cap B = \emptyset$ et si $C \subseteq B$ alors $A \cap C = \emptyset$.

Solution.

(a) atmosphère composée de 100% d'oxygène $\Rightarrow$ atmosphère irrespirable pour beaucoup d'organismes.

La principale remarque à faire ici, c'est que la transformation d'une phrase de la forme « si A alors B » n'est pas « si $A \Rightarrow B$ ». En effet, le symbole d'implication $\Rightarrow$ se lit « implique », ou encore « si ...alors ... » (et contient donc le « si »). On peut aussi remarquer qu'à cause de l'absence du « si », la phrase

> l'atmosphère était composée de 100% d'oxygène $\Rightarrow$ elle serait irrespirable pour beaucoup d'organismes

est très laide et ne rend pas bien l'idée. Finalement, le pronom personnel « elle » a été remplacé par le nom qu'il désigne.

(b) $A \cap B = \emptyset \ \wedge \ C \subseteq B \ \Rightarrow \ A \cap C = \emptyset$.

Problème 2, page 21. Soient A, B et C des ensembles quelconques. Montrez par contradiction que si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$.

Solution. Voici tout d'abord trois solutions possibles. Ensuite, nous discuterons d'autres manières de poser le problème et de présenter sa solution.

Il s'agit de montrer que $\neg P \Rightarrow$ faux, où P est la proposition : si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$.

$\neg$ (si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$)

$\Leftrightarrow$ $\langle$ Reformulation sous forme d'implication. $\rangle$

$\neg((A \cap B = \emptyset \wedge C \subseteq B) \Rightarrow A \cap C = \emptyset)$

$\Leftrightarrow$ $\langle$ Par 0.49. $\rangle$

$A \cap B = \emptyset \wedge C \subseteq B \wedge A \cap C \neq \emptyset$

$\Rightarrow$ $\langle$ $C \subseteq B \Rightarrow A \cap C \subseteq A \cap B$ $\rangle$

$A \cap B = \emptyset \wedge A \cap C \subseteq A \cap B \wedge A \cap C \neq \emptyset$

$\Rightarrow$ $\langle$ $A \cap C \subseteq A \cap B \wedge A \cap B = \emptyset \Rightarrow A \cap C = \emptyset$ $\rangle$

$A \cap C = \emptyset \wedge A \cap C \neq \emptyset$

$\Leftrightarrow$ $\langle$ Contradiction. $\rangle$

faux.

Voici une autre preuve.

$\neg$ (si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$)

$\Leftrightarrow$ $\langle$ Reformulation sous forme d'implication. $\rangle$

$\neg((A \cap B = \emptyset \wedge C \subseteq B) \Rightarrow A \cap C = \emptyset)$

$\Leftrightarrow$ $\langle$ Par 0.49. $\rangle$

$A \cap B = \emptyset \wedge C \subseteq B \wedge A \cap C \neq \emptyset$

$\Rightarrow$ $\langle$ Puisque $A \cap C \neq \emptyset$, il existe un élément x qui appartient à la fois à A et à C. $\rangle$

$A \cap B = \emptyset \wedge C \subseteq B \wedge (\exists x : x \in A \wedge x \in C)$

$\Rightarrow$ $\langle$ Puisque $C \subseteq B$, tout élément de C est élément de B. $\rangle$

$A \cap B = \emptyset \wedge (\exists x : x \in A \wedge x \in B)$

$\Rightarrow$ $\langle$ Puisque $x \in A$ et $x \in B$, on a $A \cap B \neq \emptyset$. $\rangle$

$A \cap B = \emptyset \wedge A \cap B \neq \emptyset$

$\Leftrightarrow$ $\langle$ Contradiction. $\rangle$

faux.

Finalement, voici une dernière preuve, basée sur les définitions des opérations ensemblistes. Il faut une bonne connaissance des lois des quantificateurs pour travailler à ce niveau.

$\neg$ (si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$)

$\Leftrightarrow$ $\langle$ Reformulation sous forme d'implication. $\rangle$

$\neg((A \cap B = \emptyset \wedge C \subseteq B) \Rightarrow A \cap C = \emptyset)$

$\Leftrightarrow$ $\langle$ Par 0.49. $\rangle$

$A \cap B = \emptyset \wedge C \subseteq B \wedge A \cap C \neq \emptyset$

$\Rightarrow$ $\langle$ Définition de $\cap$ et $\subseteq$. $\rangle$

$\neg(\exists x : x \in A \wedge x \in B) \wedge (\forall y : y \in C \Rightarrow y \in B) \wedge (\exists x : x \in A \wedge x \in C)$

$\Leftrightarrow$ $\langle$ x non libre dans $(\forall y : y \in C \Rightarrow y \in B)$. $\rangle$

$\neg(\exists x : x \in A \wedge x \in B) \wedge (\exists x : (\forall y : y \in C \Rightarrow y \in B) \wedge x \in A \wedge x \in C)$

$\Rightarrow$ $\langle$ $x \in C \wedge (\forall y : y \in C \Rightarrow y \in B) \Rightarrow x \in B$. $\rangle$

$\neg(\exists x : x \in A \wedge x \in B) \wedge (\exists x : x \in A \wedge x \in B)$

$\Leftrightarrow$ $\langle$ Contradiction. $\rangle$

faux.

Voici maintenant quelques commentaires sur la manière de poser le problème et de présenter sa solution.

L'énoncé du problème est le suivant : Soient A, B et C des ensembles quelconques. Montrez par contradiction que si $A \cap B = \emptyset$ et $C \subseteq B$, alors $A \cap C = \emptyset$.

Il y a plusieurs manières de traduire un tel énoncé sous forme mathématique. Tout d'abord, il faut décider quelles sont les hypothèses *globales*, c'est-à-dire celles qu'on ne traîne pas dans la formule qui est transformée dans la dérivation. Dans le cas des trois solutions précédentes, l'hypothèse globale est que A, B, C sont des ensembles. En fait, on omet souvent d'indiquer explicitement de semblables hypothèses dans la formalisation, car le fait qu'on applique les opérations ensemblistes à A, B, C nous les rappelle constamment. La formule à manipuler, qui correspond à l'énoncé, est alors $(A \cap B = \emptyset \wedge C \subseteq B) \Rightarrow A \cap C = \emptyset$ (les parenthèses pourraient être supprimées, à cause des règles de précédence des opérateurs logiques ; c'est ce qui sera fait par la suite). Nous pourrions aussi tout inclure dans la formule, qui deviendrait alors

$$\forall A, B, C \text{ ensembles} : A \cap B = \emptyset \wedge C \subseteq B \Rightarrow A \cap C = \emptyset.$$

La dérivation se poursuivrait comme dans la solution donnée. Les formules sont tout simplement un peu plus longues. Une autre traduction possible est

$$\forall A, B, C : A, B, C \text{ ensembles} \Rightarrow (A \cap B = \emptyset \wedge C \subseteq B \Rightarrow A \cap C = \emptyset),$$

ou encore

$$A, B, C \text{ ensembles} \Rightarrow (A \cap B = \emptyset \wedge C \subseteq B \Rightarrow A \cap C = \emptyset),$$

la quantification universelle étant implicite. En utilisant la loi

$$(p \Rightarrow (q \Rightarrow r)) \Leftrightarrow (p \wedge q \Rightarrow r),$$

que vous pouvez vérifier au moyen d'une table de vérité, cette dernière formulation devient

$$A, B, C \text{ ensembles} \wedge A \cap B = \emptyset \wedge C \subseteq B \Rightarrow A \cap C = \emptyset. \tag{5.1}$$

Par contre, ceci est incorrect :

$$A, B, C \text{ ensembles} \wedge (A \cap B = \emptyset \wedge C \subseteq B \Rightarrow A \cap C = \emptyset),$$

car il n'est pas vrai que tout A (par exemple) soit un ensemble. Il est aussi possible de considérer que les hypothèses globables sont

$$A, B, C \text{ sont des ensembles}, A \cap B = \emptyset \text{ et } C \subseteq B$$

(les prémisses de l'implication dans la formulation 5.1 deviennent des hypothèses globales). Il faut alors montrer $A \cap C = \emptyset$. Cela peut être fait en procédant par contradiction :

$$
\begin{array}{ll}
 & \neg(A \cap C = \emptyset) \\
\Leftrightarrow & \quad \langle \text{ Définition de } \neq. \rangle \\
 & A \cap C \neq \emptyset \\
\Rightarrow & \quad \langle \text{ Hypothèse } C \subseteq B. \rangle \\
 & A \cap B \neq \emptyset \\
\Rightarrow & \quad \langle \text{ Hypothèse } A \cap B = \emptyset. \rangle \\
 & \mathsf{faux}.
\end{array}
$$

Remarquez comment la formule d'une ligne donnée découle des hypothèses globales et de la ligne précédente.

Voici maintenant un exemple de dérivation incorrecte :

$$
\begin{array}{ll}
 & A \cap B = \emptyset \wedge C \subseteq B \\
\Rightarrow & \quad \langle \text{ Par la loi } p \wedge q \Rightarrow p. \rangle \\
 & A \cap B = \emptyset \\
\Rightarrow & \quad \langle \text{ Par la loi } p \wedge q \Rightarrow q. \rangle \\
 & C \subseteq B.
\end{array}
$$

La dernière ligne découle de la première, mais il est incorrect d'utiliser une ligne qui n'est pas la ligne précédente. C'est que la dérivation précédente est équivalente à

$$A \cap B = \emptyset \ \wedge \ C \subseteq B \ \Rightarrow \ A \cap B = \emptyset$$

et

$$A \cap B = \emptyset \ \Rightarrow \ C \subseteq B.$$

Or cette dernière implication est fausse.

Problème 3, page 21. Montrez par induction que si $x \neq y$, alors $x^n - y^n$ est divisible par $x - y$, pour tout n dans $\mathbf{N}^+$.

Solution. Nous voulons montrer que si $x \neq y$, alors

$$\forall n \in \mathbf{N}^+ : x^n - y^n \text{ est divisible par } x - y.$$

Le prédicat d'induction $P(n)$ est

$$x^n - y^n \text{ est divisible par } x - y.$$

On remarque qu'il y a un problème lorsque $x = y$, puisqu'alors le prédicat d'induction est équivalent à « 0 est divisible par 0 ». On peut décider d'exclure ce cas (ce que nous faisons ici, puisque l'énoncé précise que $x \neq y$), ou bien décider que $0/0 = 1$.

- Base d'induction : $n = 1$ (c'est le plus petit élément de $\mathbf{N}^+$). Puisque
  $x^1 - y^1 = x - y$,
  et que $x - y$ est divisible par $x - y$, on voit que $x^1 - y^1$ est divisible par $x - y$.
- Étape d'induction : Montrons que pour tout $k \in \mathbf{N}^+$, $P(k) \Rightarrow P(k+1)$.

  $P(k)$

  $\Leftrightarrow$ $\langle$ Définition de P. $\rangle$

  $x^k - y^k$ est divisible par $x - y$

  $\Rightarrow$ $\langle$ Donc $x(x^k - y^k)$ est divisible par $x - y$; mais $y^k(x - y)$ l'est aussi. $\rangle$

  $x(x^k - y^k) + y^k(x - y)$ est divisible par $x - y$

  $\Leftrightarrow$ $\langle$ Lois des polynômes : $x(x^k - y^k) + y^k(x - y) = x^{k+1} - y^{k+1}$. $\rangle$

  $x^{k+1} - y^{k+1}$ est divisible par $x - y$

  $\Leftrightarrow$ $\langle$ Définition de P. $\rangle$

  $P(k+1)$.

  Une autre manière de démontrer $P(k) \Rightarrow P(k+1)$ est la suivante.
  Supposons que $P(k)$ est vrai (hypothèse d'induction), c'est-à-dire que
  $x^k - y^k$ est divisible par $x - y$.
  Montrons que cela entraîne la vérité de $P(k+1)$, qui est
  $x^{k+1} - y^{k+1}$ est divisible par $x - y$.
  D'après les lois des polynômes,
  $x^{k+1} - y^{k+1} = x(x^k - y^k) + y^k(x - y)$.
  Le terme $x(x^k - y^k)$ est divisible par $x - y$, à cause de l'hypothèse d'induction. Le terme $y^n(x - y)$ est trivialement divisible par $x - y$. La somme de ces deux termes, c'est-à-dire $x^{k+1} - y^{k+1}$, est donc divisible par $x - y$.

Remarquons que c'est une erreur de dire que « $x^n - y^n$ divisible par $x - y$ » est équivalent à « $x^n - y^n = k(x - y)$, où k est un entier ». Il n'y a rien dans l'énoncé qui indique que x et y sont des variables entières. Il faut donc considérer x^n, y^n et k comme des polynômes. Même si l'énoncé indiquait que les polynômes s'évaluent sur les entiers, la formulation « $x^n - y^n = k(x-y)$, où k est un entier » est incorrecte. En effet, on a $2^2 - 1^2 = 3 \times (2-1)$, d'où $k = 3$, mais on a aussi $3^2 - 1^2 = 4 \times (3-1)$, d'où $k = 4$, ce qui est contradictoire. La formulation correcte est « $\exists k \in \mathbf{Z} : x^n - y^n = k(x - y)$ ».

Voici quelques commentaires sur ce problème.

- Comme chacun le sait, $\frac{x^n - y^n}{x - y}$ est le résultat de la division de $x^n - y^n$ par $x - y$. Comme la solution du problème le montre, ce résultat est un polynôme. Il est donc incorrect d'écrire $P(n) = \frac{x^n - y^n}{x - y}$, puisqu'un prédicat ne peut être égal à un polynôme. Par contre, on peut écrire $P(n) \Leftrightarrow (x - y) | (x^n - y^n)$, car la notation $a|b$ signifie que a divise b.

- Il est incorrect de définir P par

  $P(n) \Leftrightarrow x^n - y^n$ est divisible par $x - y$ pour tout $n \in \mathbf{N}^+$,

  ou, de manière équivalente, par

  $P(n) \Leftrightarrow (\forall n \in \mathbf{N}^+ : x^n - y^n$ est divisible par $x - y)$.

  D'une part, $(\forall n \in \mathbf{N}^+ : x^n - y^n$ est divisible par $x - y)$ est la proposition à démontrer. D'autre part, le quantificateur universel lie la variable n, qui devient locale à la formule quantifiée (c'est exactement comme une déclaration de variable locale dans un langage de programmation). La variable globale n dans $\ll P(n) \gg$ ne réfère donc plus à la variable n qui est quantifiée.
- Dans ce problème, on demande de montrer que si $x \neq y$, alors

  $\forall n \in \mathbf{N}^+ : x^n - y^n$ est divisible par $x - y$.

  Étant donné que

  $x^n - y^n$ divisible par $x - y \Leftrightarrow (\exists q : x^n - y^n = q(x - y))$,

  on peut donner une autre démonstration en utilisant le prédicat d'induction $P(n)$ suivant :

  $(\exists q : x^n - y^n = q(x - y))$.

  On remarque qu'avec cette formulation, il n'y a plus de problème lorsque $x = y$. C'est la raison pour laquelle cette hypothèse ne sera pas utilisée dans ce qui suit.
  - Base d'induction : $n = 1$ (c'est le plus petit élément de $\mathbf{N}^+$).

    $P(1)$

    $\Leftrightarrow$ $\langle$ Définition de P. $\rangle$

    $(\exists q : x^1 - y^1 = q(x - y))$

    $\Leftarrow$ $\langle$ Remarquez le sens de l'implication : elle pointe vers $P(1)$, qu'on veut prouver. $\rangle$

    $x^1 - y^1 = 1(x - y)$

    $\Leftrightarrow$ vrai
  - Étape d'induction : Montrons que pour tout $k \in \mathbf{N}^+$, $P(k) \Rightarrow P(k+1)$.

    $P(k)$

    $\Leftrightarrow$ $\langle$ Définition de P. $\rangle$

    $(\exists q : x^k - y^k = q(x - y))$

    $\Rightarrow$ $\langle$ Multiplication des deux côtés de l'égalité par x. $\rangle$

    $(\exists q : x(x^k - y^k) = xq(x - y))$

    $\Leftrightarrow$ $\langle$ Ajout d'un même terme de chaque côté de l'égalité. $\rangle$

    $(\exists q : x(x^k - y^k) + (x - y)y^k = xq(x - y) + (x - y)y^k)$

    $\Leftrightarrow$ $\langle$ Un peu d'algèbre. $\rangle$

    $(\exists q : x^{k+1} - y^{k+1} = (xq + y^k)(x - y))$

$$
\begin{aligned}
&\Rightarrow (\exists r : x^{k+1} - y^{k+1} = r(x-y)) \\
&\Leftrightarrow \qquad \langle \text{ Définition de } P. \rangle \\
&\quad P(k+1)
\end{aligned}
$$

Problème 4, page 21. Montrez par induction que si x est un nombre non négatif, alors $(1+x)^n - 1 \geq nx$ pour tout n dans $\mathbf{N}$.

Solution. Appelons $P(n)$ le prédicat $(1+x)^n - 1 \geq nx$. Il faut montrer qu'il est vrai pour tout $n \in \mathbf{N}$.

- Base d'induction : n = 0 (c'est le plus petit élément de $\mathbf{N}$). Le prédicat $P(0)$ est vrai, car
  $(1+x)^0 - 1 = 1 - 1 = 0 \geq 0x$.
- Étape d'induction. Il faut montrer que $P(k) \Rightarrow P(k+1)$, quel que soit k.

  $$
  \begin{aligned}
  &P(k) \\
  \Leftrightarrow &\qquad \langle \text{ Définition de } P(k). \rangle \\
  &(1+x)^k - 1 \geq kx \\
  \Rightarrow &\qquad \langle \text{ Multiplication à gauche par } (1+x), \text{ qui est} \geq 1. \rangle \\
  &(1+x)[(1+x)^k - 1] \geq kx \\
  \Leftrightarrow &\qquad \langle \text{ Distributivité de } \times \text{ sur } - \text{ et définition de puissance. } \rangle \\
  &(1+x)^{k+1} - (1+x) \geq kx \\
  \Leftrightarrow &\qquad \langle \text{ Transfert du } x \text{ de gauche à droite. } \rangle \\
  &(1+x)^{k+1} - 1 \geq (k+1)x \\
  \Leftrightarrow &\qquad \langle \text{ Définition de } P(k+1). \rangle \\
  &P(k+1).
  \end{aligned}
  $$

  Une autre façon de faire l'étape d'induction est la suivante. Prenons $P(k)$ comme hypothèse d'induction ; i.e., supposons que $(1+x)^k - 1 \geq kx$ soit vrai. Il faut montrer qu'alors $P(k+1)$ est vrai, c'est-à-dire que $(1+x)^{k+1} - 1 \geq (k+1)x$ (c'est tout simplement une autre façon de prouver $P(k) \Rightarrow P(k+1)$).

  $$
  \begin{aligned}
  &(1+x)^{k+1} - 1 \\
  = &\ (1+x)(1+x)^k - 1 \\
  = &\qquad \langle \text{ Distributivité. } \rangle \\
  &x(1+x)^k + (1+x)^k - 1 \\
  \geq &\qquad \langle \text{ Par l'hypothèse d'induction. } \rangle \\
  &x(1+x)^k + kx \\
  \geq &\qquad \langle \text{ Puisque } x \text{ est non négatif, } (1+x)^k \geq 1. \rangle
  \end{aligned}
  $$

$$
\begin{aligned}
& x + kx \\
= \; & (k+1)x.
\end{aligned}
$$

À cause de la transitivité de $\geq$, cette preuve montre que $P(k+1)$ est vrai.

Problème 5, page 21. Montrez par induction que $n! > 2^n$, pour tout $n \geq 4$.

Solution. Soit $P(n)$ le prédicat défini par $P(n) \Leftrightarrow n! > 2^n$. Il faut montrer qu'il est vrai pour tout $n \geq 4$.

- Base d'induction : n = 4. Le prédicat $P(4)$ est vrai, car
  $4! = 1 \times 2 \times 3 \times 4 = 24 > 16 = 2^4$.
- Étape d'induction. Il faut montrer $P(k) \Rightarrow P(k+1)$, pour tout $k \geq 4$.

$$
\begin{aligned}
& P(k) \\
\Leftrightarrow \quad & \langle \text{ Définition de } P. \rangle \\
& k! > 2^k \\
\Rightarrow \quad & \langle \; k+1 > 2, \text{ puisque } k \geq 4. \; \rangle \\
& k!(k+1) > 2 \times 2^k \\
\Leftrightarrow \quad & \langle \text{ Définition de factorielle et de puissance. } \rangle \\
& (k+1)! > 2^{k+1} \\
\Leftrightarrow \quad & \langle \text{ Définition de } P. \rangle \\
& P(k+1).
\end{aligned}
$$

L'étape d'induction peut aussi se démontrer de la manière suivante. Prenons $P(k)$ comme hypothèse d'induction, c'est-à-dire supposons $k! > 2^k$. Il faut montrer $P(k+1)$, c'est-à-dire $(k+1)! > 2^{k+1}$ (c'est une autre façon de prouver $P(k) \Rightarrow P(k+1)$).

$$
\begin{aligned}
& (k+1)! \\
= \quad & \langle \text{ Définition de factorielle. } \rangle \\
& k!(k+1) \\
> \quad & \langle \text{ Par l'hypothèse d'induction, } k! > 2^k. \; \rangle \\
& 2^k(k+1) \\
> \quad & \langle \text{ Puisque } k \geq 4, \text{ on a } k+1 > 2. \; \rangle \\
& 2^k \times 2 \\
= \quad & \langle \text{ Définition de puissance. } \rangle \\
& 2^{k+1}
\end{aligned}
$$

Problème 1, page 30.

(a) Comparez les cardinalités des ensembles suivants et justifiez votre réponse.

$$P = \{k \in \mathbf{N} : k < 15 \wedge (\exists j \in \mathbf{N} : k = 2j)\}$$
$$Q = \{k \in \mathbf{N} : k < 15 \wedge (\exists j \in \mathbf{N} : k = 3j)\}$$

Solution. Il est facile de voir que

$$P = \{0, 2, 4, 6, 8, 10, 12, 14\},$$
$$Q = \{0, 3, 6, 9, 12\}.$$

La manière la plus simple de comparer les cardinalités, c'est de passer par l'intermédiaire des ensembles C_n (notation 0.54). Bien sûr, la réponse est évidente ; il s'agit ici de pratiquer l'usage des concepts sur un cas simple. On a $|C_8| = |P|$, car la fonction $f : C_8 \to P$ définie par

$$f = \{(0,0), (1,2), (2,4), (3,6), (4,8), (5,10), (6,12), (7,14)\}$$

est bijective. De même, $|C_5| = |Q|$, car la fonction $g : C_5 \to Q$ définie par

$$g = \{(0,0), (1,3), (2,6), (3,9), (4,12)\}$$

est bijective. Par conséquent, en utilisant la définition 0.56, nous obtenons

$$|Q| = |C_5| = 5 < 8 = |C_8| = |P|,$$

c'est-à-dire $|Q| < |P|$.

Au lieu d'utiliser les ensembles C_n, on peut aussi remarquer que la fonction $h : Q \to P$ définie par

$$h = \{(0,0), (3,2), (6,4), (9,6), (12,8)\}$$

est injective, d'où $|Q] \leq |P]$. On invoque ensuite le fait qu'il n'y a pas de fonction surjective de Q dans P ; toutefois, il est difficile de justifier très rigoureusement cette assertion, à moins d'énumérer toutes les fonctions de Q dans P.

(b) Quelle est la cardinalité des ensembles suivants ? Justifiez votre réponse.

i. $\{a\}$

Solution. $|\{a\}| = 1$, car la fonction $f : C_1 \to \{a\}$ définie par $f = \{(0, a)\}$ est bijective. Par conséquent, par la définition 0.56, $|\{a\}| = |C_1| = 1$.

ii. $\emptyset$

Solution. $|\emptyset| = 0$, car la fonction $f : C_0 \to \emptyset$ définie par $f = \emptyset$ est bijective. Par conséquent, par la définition 0.56, $|\emptyset| = |C_0| = 0$.

(c) Soient les ensembles $A = \{\{0,1\}, \{a\}\}, B = \{\alpha, \beta, \gamma\}$ et $C = \{0, 1, 2, 3, 4\}$. Quelle est la cardinalité des ensembles suivants ?

i. $B \times C$

ii. $C \times B$

iii. $A \times B \times C$

iv. $C \times \emptyset$

Solution. Le plus simple est d'utiliser la propriété $|A \times B| = |A| \times |B|$ (voyez à la page 24), en assumant $|A| = 2, |B| = 3$ et $|C| = 5$. On obtient alors les cardinalités suivantes.

i. $B \times C = |B| \times |C| = 3 \times 5 = 15$

ii. $C \times B = |C| \times |B| = 5 \times 3 = 15$

iii. $A \times B \times C = |A| \times |B| \times |C| = 2 \times 3 \times 5 = 30$

iv. $C \times \emptyset = |C| \times |\emptyset| = 5 \times 0 = 0$

Il est aussi possible de développer les produits cartésiens et de comparer les ensembles résultants aux ensembles C_n appropriés pour obtenir leur cardinalité.

Problème 2, page 30. Donnez un exemple de deux ensembles A et B, *non vides* et *disjoints*, tels que

(a) $|A| < |B| < |A \cup B|$; **solution** : $A = \{0\}, B = \{1, 2\}$, d'où $A \cup B = \{0, 1, 2\}$.

(b) $|A| < |B| = |A \cup B|$; **solution** : $A = \{0\}, B = \mathbf{N}^+$, d'où $A \cup B = \mathbf{N}$.

(c) $|A| = |B| = |A \cup B|$; **solution** : $A = \{k : (\exists j \in \mathbf{N} : k = 2j)\}, B = \{k : (\exists j \in \mathbf{N} : k = 2j+1)\}$, d'où $A \cup B = \mathbf{N}$. Autrement dit, A est l'ensemble des naturels pairs et B l'ensemble des naturels impairs.

Problème 3, page 30. Montrez que $|\mathbf{N}| = |\mathbf{N} \times \mathbf{N} \times \mathbf{N}|$.

Solution. Tout d'abord, la partie triviale : $|\mathbf{N}| \leq |\mathbf{N} \times \mathbf{N} \times \mathbf{N}|$. En effet, la fonction $f : \mathbf{N} \to \mathbf{N} \times \mathbf{N} \times \mathbf{N}$ définie par

$$f(n) = (n, 0, 0)$$

est injective, puisque $j \neq k \Rightarrow f(j) \neq f(k)$.

Il reste à montrer que $|\mathbf{N} \times \mathbf{N} \times \mathbf{N}| \leq |\mathbf{N}|$, c'est-à-dire que $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$ est dénombrable.

Pour énumérer les triplets de $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$, on commence par énumérer ceux dont la somme est 0 (il n'y en a qu'un : (0,0,0)), puis ceux dont la somme est 1 (ce sont : (0,0,1),(0,1,0),(1,0,0)), ..., puis ceux dont la somme est n. Comme il y a un nombre fini de triplets dont la somme est n, cette énumération se fait en un nombre fini d'étapes. On peut ensuite énumérer les triplets dont la somme est $n + 1$, etc. Il

est évident qu'avec une telle procédure, n'importe quel triplet de $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$ est éventuellement nommé.

Cet argument est suffisant et vous n'aviez pas à fournir une preuve plus élaborée. Toutefois, il est aussi possible de montrer qu'il existe une fonction injective $f : \mathbf{N} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$, par exemple la fonction f définie par

$$f(a,b,c) = \frac{(a+b+c)(a+b+c+1)(a+b+c+2)}{6} + \frac{(a+b)(a+b+1)}{2} + a.$$

Il n'est cependant pas très facile de trouver une telle fonction, à moins d'avoir déjà vu le truc. De plus, la preuve de l'injectivité de f est un peu longue et n'est pas incluse.

Problème 4, page 30. En utilisant les définitions 0.62 et 0.64, dites si les ensembles suivants sont finis, infinis, dénombrables ou non dénombrables et expliquez pourquoi.

(a) $\emptyset$

Solution. La seule fonction de l'ensemble $\emptyset$ vers l'ensemble $\mathbf{N}$ est la fonction vide ($\emptyset$ ou $\{\}$). Elle est injective mais pas surjective. Par conséquent, par la définition 0.58, $|\emptyset| < |\mathbf{N}|$. Par la définition 0.62, l'ensemble $\emptyset$ est fini. Par la définition 0.64, il est dénombrable.

(b) $P = \{k \in \mathbf{N} : (\exists j \in \mathbf{N} : k = 2j)\}$

Solution. Par l'exercice 0.60(2), on a $|P| = |\mathbf{N}|$. Selon la définition 0.62, l'ensemble P est infini. Selon la définition 0.64, il est dénombrable.

Problème 5, page 30. Donnez des exemples qui montrent que l'intersection de deux ensembles non dénombrables peut être :

(a) finie (un exemple) ;

(b) infinie, mais dénombrable (un exemple) ;

(c) non dénombrable (un exemple).

Solution. Ce problème a bien sûr plusieurs solutions. En voici quelques-unes. Dans tous les cas, les deux ensembles non dénombrables sont identifiés par A et B. L'ensemble $\mathbf{R}$ est l'ensemble des réels. L'ensemble $\mathbf{Z}$ est l'ensemble des entiers relatifs. L'ensemble $[x,y]$ est défini par $[x,y] = \{r \in \mathbf{R} : x \leq r \leq y\}$. Remarquez la simplicité de la première réponse dans chaque catégorie.

(a)
- $A = \wp(\mathbf{N})$, $B = \mathbf{R}$, $A \cap B = \emptyset$.
- $A = \wp(\{n : n \in \mathbf{N} \wedge n \text{ est pair}\})$,
  $B = \wp(\{n : n \in \mathbf{N} \wedge n \text{ est impair}\})$,
  $A \cap B = \{\emptyset\}$,

- $A = [0,1],\ B = [2,3],\ A \cap B = \emptyset$.

(b)
- $A = \wp(\mathbf{N}) \cup \mathbf{N},\ B = \mathbf{R} \cup \mathbf{N},\ A \cap B = \mathbf{N}$.
- $A = \{(x,y) : x \in \mathbf{R} \wedge y \in \mathbf{R} \wedge y = \text{sinus}(x)\}$,
  $B = \{(x,y) : x \in \mathbf{R} \wedge y \in \mathbf{R} \wedge y = 0\}$,
  $A \cap B = \{(k\pi, 0) : k \in \mathbf{Z}\}$.
  Les ensembles A et B contiennent des paires de réels. L'ensemble $A \cap B$ contient les points communs à l'axe des x et à la courbe sinus(x).

(c)
- $A = B = \mathbf{R},\ A \cap B = \mathbf{R}$.
- $A = [0,2],\ B = [1,3],\ A \cap B = [1,2]$.
- $A = \wp(\{a,b\}^*),\ B = \wp(\{b,c\}^*),\ A \cap B = \wp(\{b\}^*)$.
  A est l'ensemble des langages sur l'alphabet $\{a,b\}$, B est l'ensemble des langages sur l'alphabet $\{b,c\}$ et $A \cap B$ est l'ensemble des langages sur l'alphabet $\{b\}$.

Problème 6, page 30. Montrez que si X est un ensemble non dénombrable et Y un ensemble dénombrable, alors $X - Y$ doit être non dénombrable.

Solution.

- Montrons d'abord que l'union de deux ensembles dénombrables A et B est un ensemble dénombrable. On peut le voir intuitivement de la manière suivante. Si on a une méthode pour énumérer les éléments de A dans l'ordre $a_0, a_1, a_2, \ldots$ et les éléments de B dans l'ordre $b_0, b_1, b_2, \ldots$, alors on peut énumérer les éléments de $A \cup B$ dans l'ordre $a_0, b_0, a_1, b_1, a_2, b_2, \ldots$
  Voici maintenant une façon plus rigoureuse de faire cette démonstration. Puisque A et B sont dénombrables, c'est qu'il existe deux fonctions injectives $f : A \to \mathbf{N}$ et $g : B \to \mathbf{N}$. Définissons une fonction $h : A \cup B \to \mathbf{N}$ de la manière suivante.
  $$h(x) = \begin{cases} 2f(x) & \text{si } x \in A \\ 2g(x)+1 & \text{si } x \in B - A \end{cases}$$
  La fonction h est injective. En effet, soient x et y tels que $x \neq y$; montrons que $h(x) \neq h(y)$:
  - si $x \in A$ et $y \in A$, on a $h(x) = 2f(x) \neq 2f(y) = h(y)$, car f est injective ;
  - si $x \in B - A$ et $y \in B - A$, on a $h(x) = 2g(x)+1 \neq 2g(y)+1 = h(y)$, car g est injective ;
  - si $x \in A$ et $y \in B - A$, on a $h(x) = 2f(x)$ et $h(y) = 2g(y)+1$, i.e. $h(x)$ est pair et $h(y)$ est impair ; donc $h(x) \neq h(y)$. Il en est de même si $x \in B - A$ et $y \in A$.
- Nous voulons montrer que si X est non dénombrable et Y dénombrable, alors $X - Y$ est non dénombrable. Faisons une preuve par contradiction.

$\neg$ (si X est non dénombrable et Y dénombrable, alors $X - Y$ est non dénombrable)

$\Leftrightarrow$ $\langle$ Reformulation avec l'implication. $\rangle$

$\neg(X$ non dénombrable $\wedge\, Y$ dénombrable $\Rightarrow X - Y$ non dénombrable)

$\Leftrightarrow$ $\langle$ Par 0.49. $\rangle$

X non dénombrable $\wedge\, Y$ dénombrable $\wedge\, X - Y$ dénombrable)

$\Rightarrow$ $\langle$ Par la preuve ci-dessus. $\rangle$

X non dénombrable $\wedge\, (X - Y) \cup Y$ dénombrable

$\Leftrightarrow$ $\langle$ $(X - Y) \cup Y = X \cup Y$. $\rangle$

X non dénombrable $\wedge\, X \cup Y$ dénombrable

$\Rightarrow$ $\langle$ $A \subseteq B \wedge B$ dénombrable $\Rightarrow A$ dénombrable. Donc X est dénombrable, puisque $X \cup Y$ est dénombrable. $\rangle$

X non dénombrable $\wedge\, X$ dénombrable

$\Leftrightarrow$ $\langle$ Contradiction. $\rangle$

faux.

Problème 7, page 30. Montrez que $|\mathbf{R}| \leq |\,[0,1]\,|$.

Solution. Selon la définition 0.58, il faut trouver une fonction injective $f : \mathbf{R} \to [0,1]$. En voici deux.

- $f(x) = \begin{cases} 1 - \frac{1}{2(1+x)} & \text{si } x > 0 \\ \frac{1}{2(1-x)} & \text{si } x \leq 0 \end{cases}$
  Les deux portions de f (c'est-à-dire pour $x > 0$ et $x \leq 0$) sont injectives. De plus, si $x \leq 0$ et $y > 0$, alors $f(x) > 1/2$ et $f(y) \leq 1/2$, d'où $f(x) \neq f(y)$. La fonction f est donc injective.
- $f(x) = (\frac{1}{\pi}\mathsf{arctg}\ x) + \frac{1}{2}$
  La fonction $\mathsf{arctg}\ x$ donne des valeurs comprises entre $-\pi/2$ et $\pi/2$. La division par π donne des valeurs entre $-1/2$ et $1/2$. L'addition de $1/2$ ramène le tout sur l'intervalle $[0,1]$. Vous pouvez vérifier intuitivement que f est injective, grâce à votre connaissance des propriétés des fonctions trigonométriques.

5.2 Chapitre 1

Problème 1, page 40. L'ensemble vide $\emptyset$ est-il un alphabet ?

Solution. Non. Un alphabet ne peut être vide (définition 1.1).

Problème 2, page 40. Décrivez en mots l'ensemble L suivant. Dites lesquelles des séquences données appartiennent à l'ensemble et expliquez pourquoi.

$$\Sigma = \{t, o\}, \quad L = \{(to)^{2n} : n \in \mathbf{N}\}$$

- *ttoo*
- *tototo*

Solution. L est l'ensemble des séquences de 0, 1 ou plusieurs *toto*. Aucune des séquences données n'appartient à L. La seule séquence de longueur 4 est *toto* et elle est différente de *ttoo*. La séquence *tototo* ne contient pas un nombre entier de *toto*.

Problème 3, page 40. Donnez une expression ensembliste décrivant l'ensemble des séquences sur l'alphabet $\{0\}$ qui contiennent un nombre premier de 0.

Solution. Une première ébauche de solution est l'expression

$$\{0^p : p \text{ est un nombre premier}\}.$$

Il faut ensuite exprimer formellement que p est premier. Il faut refléter le fait qu'un nombre premier p est supérieur à 1 (c'est plutôt une convention) et que s'il a un facteur q, alors $q = 1$ ou $q = p$. On obtient alors

$$\{0^p : p \in \mathbf{N} \ \wedge \ p \geq 2 \ \wedge \ (\forall q, r \in \mathbf{N} : p = qr \ \Rightarrow \ q = 1 \ \vee \ q = p)\}.$$

Problème 4, page 40. Dites si les affirmations suivantes sont vraies ou fausses. Expliquez votre choix.

(a) $\emptyset \not\subseteq \{\lambda\}$.
Faux. Pour tout ensemble S, $\emptyset \subseteq S$. Donc $\emptyset \subseteq \{\lambda\}$.

(b) $\lambda \in \emptyset$.
Faux. L'ensemble $\emptyset$ ne contient aucun élément.

(c) $\lambda \subseteq \lambda$.
Faux. L'expression est mal typée : la séquence λ n'est pas un ensemble et elle ne peut être incluse dans un autre ensemble (encore moins dans une séquence).

(d) $\emptyset = \{\lambda\}$.
Faux. L'ensemble $\emptyset$ ne contient aucun élément, alors que l'ensemble $\{\lambda\}$ contient l'élément λ.

(e) Si L est un langage sur Σ, alors $L \in \wp(\Sigma^*)$.
Vrai. Si L est un langage sur Σ, alors $L \subseteq \Sigma^*$ (définition 1.20), ce qui est équivalent à $L \in \wp(\Sigma^*)$, par définition de l'ensemble puissance d'un ensemble (définition 0.13).

(f) La chaîne $\lambda a \lambda \lambda a$ est un élément du langage $\{a, aa, aaa\}$.
Vrai, car $\lambda a \lambda \lambda a = aa$.

(g) Si un langage a pour représentation ensembliste $\{\lambda, \lambda\lambda, \lambda\lambda\lambda\}$ alors cet ensemble contient trois séquences de symboles.
Faux, car $\lambda = \lambda\lambda = \lambda\lambda\lambda$ et $\{\lambda, \lambda, \lambda\} = \{\lambda\}$.

(h) Σ^* est un langage, pour tout alphabet Σ.
Vrai, car $\Sigma^* \subseteq \Sigma^*$ (regardez la définition de langage (1.20)).

(i) Soit L un langage. Alors $\lambda \in L$.
Faux. Il se peut que $\lambda \in L$, mais pas forcément. Par exemple, le langage $\{a, ab\}$ ne contient pas λ.

(j) $\Sigma^* - \Sigma^*$ est un langage.
Vrai, car $\Sigma^* - \Sigma^* = \emptyset \subseteq \Sigma^*$ (voyez la définition 1.20).

Problème 5, page 41. Construisez l'analyseur lexical correspondant au diagramme de transitions de l'exemple 1.32.

Solution. Construisons d'abord la table de transitions associée au diagramme.

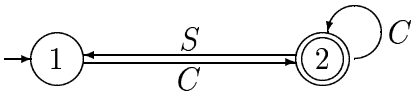
	a	b	FIN
1	2	3	ERREUR
2	3	1	ERREUR
3	3	3	ACCEPTE

Remarquons que nous avons pu le faire parce que le diagramme n'est pas ambigu (voir définition 1.34), de sorte que chaque entrée de la table contient seulement une valeur. Voici l'analyseur lexical.

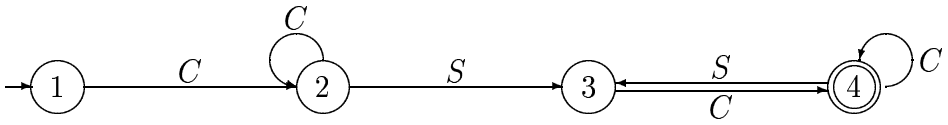
```
État := 1;
Répéter
    Lire le symbole d'entrée suivant et le mettre dans la variable SYMBOLE;
    Si SYMBOLE ∈ {a, b, FIN}
        alors État := table[État, SYMBOLE]
        sinon sortir du programme;
    Si État = "ERREUR" alors sortir du programme;
jusqu'à ce que État = "ACCEPTE".
```

Problème 6, page 41. Construisez un diagramme de transitions qui reconnaît des expressions arithmétiques de longueur arbitraire. Les expressions sont constituées d'entiers positifs (sans le symbole du signe) séparés par les symboles d'addition, de soustraction, de multiplication ou de division.

Solution. Posons $C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ et $S = \{+, -, \times, \div\}$. Il n'est pas clair à partir de la question si une expression arithmétique doit contenir au moins un entier ou si elle peut être vide. Cependant, notre connaissance de ce qu'est une expression arithmétique nous incline à supposer qu'une expression ne peut être vide. Dans ce cas, le diagramme de transitions suivant fait l'affaire.



Si la question précisait qu'il y a au moins deux entiers dans une expression, une réponse possible serait la suivante :



Problème 7, page 41. À partir du diagramme précédemment construit, donnez la table de transitions et son analyseur lexical.

Solution.

Nous utiliserons le premier diagramme. La table de transitions associée est la suivante :

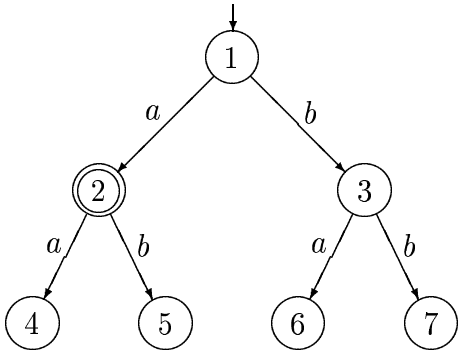
	C	S	FIN
1	2	ERREUR	ERREUR
2	2	1	ACCEPTE

Nous avons pu donner cette table parce que le diagramme n'est pas ambigu (voir définition 1.34), de sorte que chaque entrée de la table contient seulement une valeur. Voici l'analyseur lexical.

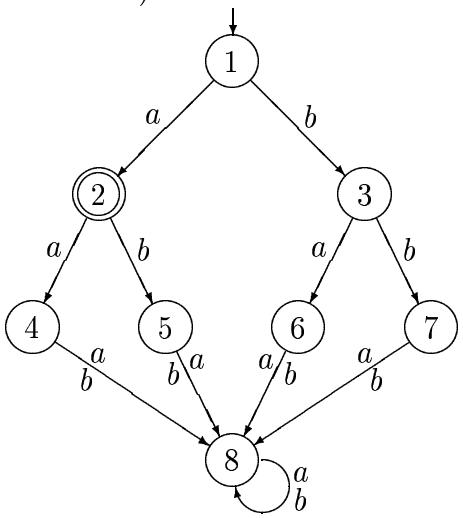
```
État := 1 ;
Répéter
    Lire le symbole d'entrée suivant et le mettre dans la variable SYMBOLE ;
    Dans le cas où SYMBOLE est
        chiffre : Entrée := "C" ;
        signe : Entrée := "S" ;
        FIN : Entrée := "FIN" ;
```

Autre : sortir du programme ;
État := table[État, Entrée] ;
Si État = “ERREUR” alors sortir du programme ;
jusqu'à ce que État = “ACCEPTE”.

Problème 1, page 48. Si possible, rendez déterministe le diagramme ci-dessous. L'alphabet est $\Sigma = \{a, b\}$.



Solution. Le diagramme n'est pas ambigu. Il est donc possible de le rendre complètement défini (et donc déterministe).

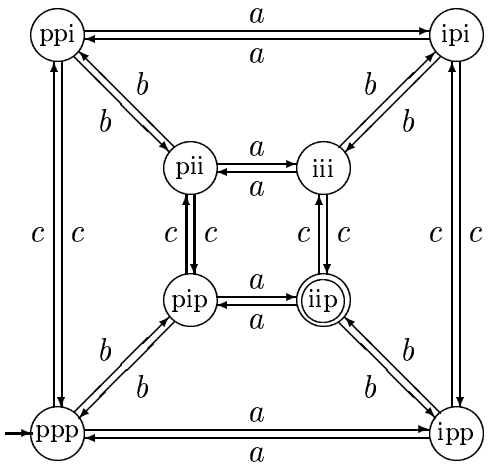


Problème 1, page 61. Montrez que l'ensemble des séquences sur l'alphabet $\{a, b, c\}$ contenant un nombre impair de a, un nombre impair de b et un nombre pair de c est un langage régulier.

Solution. Pour montrer que ce langage est régulier, nous allons construire le diagramme de transitions d'un automate fini qui le reconnaît. En tout temps lors de la lecture de la chaîne d'entrée, l'automate doit connaître la parité de a, b et c. Il doit donc distinguer huit cas (i.e. l'automate peut être dans l'un de huit états). Voici la liste des huit cas avec les noms que nous donnerons aux états correspondants.

a	b	c	états
pair	pair	pair	ppp
pair	pair	impair	ppi
pair	impair	pair	pip
pair	impair	impair	pii
impair	pair	pair	ipp
impair	pair	impair	ipi
impair	impair	pair	iip
impair	impair	impair	iii

Supposons (par exemple) que l'automate soit dans l'état pii. S'il lit un b, il doit passer dans l'état ppi, puisque la parité de b change. L'état initial est ppp, car initialement le nombre de a est pair, le nombre de b est pair et le nombre de c est pair. L'état final est iip, bien sûr.

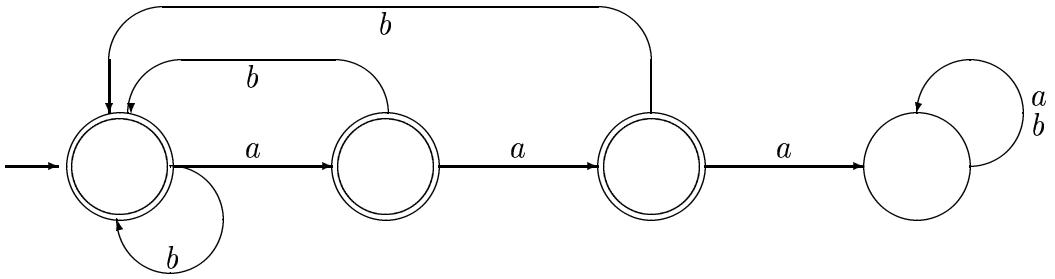


Problème 2, page 61. Montrez la régularité du langage sur $\{a, b\}$ dont les éléments sont exactement les séquences qui ne contiennent pas trois a consécutifs.

Solution. Ici aussi, exhibons le diagramme de transitions de l'automate reconnaissant le langage. Il suffit de remarquer que

1. λ doit être acceptée. L'état initial doit donc être un état final.

2. Si la partie qui a été lue est acceptable et qu'un b est lu, alors il suffit de retourner dans l'état initial.
3. Si trois a consécutifs sont détectés, il suffit d'ajouter une transition vers un état poubelle.



Problème 3, page 61. Montrez que le langage $\{a^n : n \text{ est premier}\}$ n'est pas régulier.

Solution. Faisons une preuve par contradiction.

L régulier

$\Rightarrow$ $\langle$ Définition 1.57. $\rangle$

$\exists M : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L$

$\Rightarrow$ $\langle$ Parce que L est infini —il y a en effet une infinité de nombres premiers, comme l'a démontré Euclide il y a plus de 2000 ans— il est possible de trouver un nombre premier plus grand que n'importe quel entier donné. Notez que $a^p \in L$ exprime le fait que p est premier, par définitin de L. $\rangle$

$\exists M : \exists p \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ p > |S| \ \wedge \ a^p \in L$

$\Rightarrow$ $\langle$ Puisqu'il y a plus de $|S|$ symboles a à lire, il doit y avoir une boucle de longueur $j > 0$. Cette boucle peut être traversée p fois de plus. $\rangle$

$\exists M : \exists j, p \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ p > |S| \ \wedge \ a^p \in L$
$\wedge \ j > 0 \ \wedge \ a^{p+pj} \in L(M)$

$\Rightarrow$ $\langle$ Mais $p+pj = p(j+1)$ n'est pas un nombre premier lorsque $j > 0$; en effet, $p > 1$, car c'est un nombre premier (puisque $a^p \in L$), et $j + 1 > 1$, car $j > 0$. Par conséquent, $a^{p+pj} \notin L$. $\rangle$

$\exists M : \exists j, p \in \mathbf{N} : L(M) = L \ \wedge \ a^{p+pj} \in L(M) \ \wedge \ a^{p+pj} \notin L$

$\Rightarrow$ $\langle$ $a^{p+pj} \in L(M) \ \wedge \ a^{p+pj} \notin L \ \Rightarrow \ L(M) \neq L$ $\rangle$

$\exists M : L(M) = L \ \wedge \ L(M) \neq L$

$\Leftrightarrow$ $\langle$ $(q \wedge \neg q) \Leftrightarrow$ faux et $(\exists x :$ faux$) \Leftrightarrow$ faux. $\rangle$

faux.

Voici maintenant une autre preuve, donnée dans un format plus « classique ». Elle est tout aussi bonne que l'autre.

Procédons par contradiction et supposons que ce langage soit régulier. Par définition, cela veut dire qu'il existe un automate fini M qui le reconnaît. Soit m le nombre d'états de cet automate. On sait que l'ensemble des nombres premiers est infini ; par conséquent, il existe un nombre premier plus grand que m. Soit donc p premier, $p > m$. Comme p est premier, l'automate M accepte a^p. Parce que $p > m$, il existe un état, disons s, dans lequel l'automate se trouve au moins deux fois lors de la lecture de a^p. C'est-à-dire que M peut boucler en quittant s et en y revenant un nombre arbitraire de fois avant de finalement quitter s vers l'état final. Soit $j > 0$ le nombre de a que M doit lire pour quitter s et y revenir. En bouclant une fois de plus (par rapport au nombre de boucles nécessaires pour accepter a^p), M accepte $a^p a^j$; en bouclant 2 fois de plus, il accepte $a^p a^{2j}$, etc. En bouclant p fois de plus, il accepte $a^p a^{pj} = a^{p(1+j)}$. Mais $p(1+j)$ n'est pas un nombre premier. L'automate M accepte donc une séquence qui n'appartient pas au langage $\{a^n : n \text{ est premier }\}$. Ceci est contraire à la supposition originale que M reconnaît ce langage. Il s'ensuit que la supposition est incorrecte et que ce langage n'est pas régulier.

Problème 4, page 61. Montrez que le langage $L = \{a^n b^n c^n : n \in \mathbf{N}\}$ n'est pas régulier.

Solution. Procédons par contradiction.

L régulier

$\Rightarrow$ $\langle$ Définition 1.57. $\rangle$

$\exists M : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L$

$\Rightarrow$ $\langle$ Parce que L est infini, il contient des séquences arbitrairement longues. $\rangle$

$\exists M : \exists k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ k > |S| \ \wedge \ a^k b^k c^k \in L(M)$

$\Rightarrow$ $\langle$ Puisqu'il y a plus de $|S|$ symboles a à lire, il doit y avoir une boucle de longueur $j > 0$ lors de la lecture des a. Cette boucle peut être traversée une fois de plus. $\rangle$

$\exists M : \exists j, k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ k > |S|$
$\wedge \ a^k b^k c^k \in L(M) \ \wedge \ j > 0 \ \wedge \ a^{k+j} b^k c^k \in L(M)$

$\Rightarrow$ $\langle$ $j > 0 \Rightarrow k + j \neq k$. Par conséquent, $a^{k+j} b^k c^k \notin L$. $\rangle$

$$\begin{aligned}
&\exists M : \exists j, k \in \mathbf{N} : L(M) = L \ \wedge \ a^{k+j}b^kc^k \in L(M) \ \wedge \ a^{k+j}b^kc^k \notin L \\
\Rightarrow \quad &\langle \ a^{k+j}b^kc^k \in L(M) \ \wedge \ a^{k+j}b^kc^k \notin L \ \Rightarrow \ L(M) \neq L \ \rangle \\
&\exists M : L(M) = L \ \wedge \ L(M) \neq L \\
\Leftrightarrow \quad &\langle \ (p \wedge \neg p) \Leftrightarrow \mathsf{faux} \text{ et } (\exists x : \mathsf{faux}) \Leftrightarrow \mathsf{faux}. \ \rangle \\
&\mathsf{faux}.
\end{aligned}$$

Problème 5, page 61. Montrez que le langage L ayant pour alphabet $\{a, b, c\}$ et contenant exactement tous les palindromes n'est pas régulier. Rappelons qu'un palindrome est une séquence de symboles $x_1x_2 \ldots x_n$ telle que $x_1x_2 \ldots x_n = x_nx_{n-1} \ldots x_1$. Par exemple, *abba* et *abcba* sont des palindromes.

Par contre, si L est un langage régulier, est-ce que la collection des palindromes de L est un langage régulier ? Pourquoi ?

Solution. Soit une séquence $u \in \{a, b, c\}^*$. La notation $\widehat{u}$ désigne la séquence u à l'envers. Par exemple, $\widehat{abac} = caba$. La séquence $u\widehat{u}$ est donc un palindrome. Rappelons que si S est un ensemble et u une séquence, alors $|S|$ désigne la cardinalité de S et $|u|$ la longueur de la séquence u. La preuve qui suit procède par contradiction.

$$\begin{aligned}
&L \text{ régulier} \\
\Rightarrow \quad &\langle \text{ Définition 1.57. } \rangle \\
&\exists M : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \\
\Rightarrow \quad &\langle \text{ Parce que } L \text{ est infini, il contient des séquences arbitrairement longues. } \rangle \\
&\exists M : \exists u \in \{a, b, c\}^* : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ |u| > |S| \\
&\qquad\qquad \wedge \ u\widehat{u} \in L(M) \\
\Rightarrow \quad &\langle \text{ Puisqu'il y a plus de symboles dans } u \text{ que d'états dans } S, \text{ il doit y avoir une boucle de longueur supérieure à zéro lors de la lecture de } u. \text{ La séquence } u \text{ a alors la forme } u_1u_2u_3, \text{ où } u_2 \text{ est la séquence (non vide) lue dans la boucle. Cette boucle peut être traversée une fois de plus. } \rangle \\
&\exists M : \exists u, u_1, u_2, u_3 \in \{a, b, c\}^* : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \\
&\qquad\qquad \wedge \ u\widehat{u} \in L(M) \ \wedge \ u = u_1u_2u_3 \ \wedge \ |u_2| > 0 \\
&\qquad\qquad \wedge \ u_1u_2u_2u_3\widehat{u} \in L(M) \\
\Rightarrow \quad &\langle \text{ Puisque } |u_2| > 0, \text{ on a } u \neq u_1u_2u_2u_3, \text{ d'où } u_1u_2u_2u_3\widehat{u} \notin L. \ \rangle \\
&\exists M : \exists u, u_1, u_2, u_3 \in \{a, b, c\}^* : u_1u_2u_2u_3\widehat{u} \in L(M) \ \wedge \ u_1u_2u_2u_3\widehat{u} \notin L \\
\Rightarrow \quad &\langle \ u_1u_2u_2u_3\widehat{u} \in L(M) \ \wedge \ u_1u_2u_2u_3\widehat{u} \notin L \ \Rightarrow \ L(M) \neq L \ \rangle
\end{aligned}$$

$$\begin{array}{ll} & \exists M : \exists u, u_1, u_2, u_3 \in \{a,b,c\}^* : L(M) = L \ \wedge \ L(M) \neq L \\ \Leftrightarrow & \qquad \langle\ (p \wedge \neg p) \Leftrightarrow \mathsf{faux} \text{ et } (\exists x : \mathsf{faux}) \Leftrightarrow \mathsf{faux}.\ \rangle \\ & \mathsf{faux}. \end{array}$$

Pour ce qui est de la deuxième partie de la question, la réponse est négative. Par exemple, $\{a,b,c\}^*$ est un langage régulier. Or, nous venons de démontrer que l'ensemble des palindromes de $\{a,b,c\}^*$ n'est pas un langage régulier.

Problème 6, page 61. Soit L le langage avec alphabet $\Sigma = \{a,b\}$ dont les séquences ont le même nombre de a que de b. Montrez que L n'est pas régulier.

Solution. La solution de ce problème est essentiellement la même que celle de l'exercice 1.63 et est basée sur le théorème 1.61. Supposons qu'il existe un automate M tel que $L(M) = L$. Le langage L contient des séquences de la forme $a^n b^n$ pour des n arbitrairement grands (ces séquences sont un cas particulier de séquences ayant un nombre égal de a et de b) et $L(M)$ doit donc les contenir aussi. Puisque $L(M)$ est régulier (par la définition 1.57), il contient aussi des séquences de la forme $a^m b^n$ avec $m \neq n$ (par le théorème 1.61). Mais $a^m b^n \notin L$ et donc $L(M) \neq L$. À cause de cette contradiction, nous concluons qu'il n'existe pas d'automate M tel que $L(M) = L$.

Il est aussi possible de donner une preuve par contradiction semblable à celle du problème 4. Voici une telle preuve.

$$\begin{array}{ll} & L \text{ régulier} \\ \Rightarrow & \qquad \langle\ \text{Définition 1.57.}\ \rangle \\ & \exists M : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \\ \Rightarrow & \qquad \langle\ L \text{ contient des séquences arbitrairement longues de la forme } a^n b^n \\ & \qquad \text{(cas particulier de séquences ayant un nombre égal de } a \text{ et de } b).\ \rangle \\ & \exists M : \exists k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ k > |S| \ \wedge \ a^k b^k \in L(M) \\ \Rightarrow & \qquad \langle\ \text{Puisqu'il y a plus de } |S| \text{ symboles } a \text{ à lire, il doit y avoir une} \\ & \qquad \text{boucle de longueur } j > 0 \text{ lors de la lecture des } a. \text{ Cette boucle peut} \\ & \qquad \text{être traversée une fois de plus.}\ \rangle \\ & \exists M : \exists j, k \in \mathbf{N} : M = (S, \Sigma, \delta, \iota, F) \ \wedge \ L(M) = L \ \wedge \ k > |S| \\ & \qquad\qquad \wedge \ a^k b^k \in L(M) \ \wedge \ j > 0 \ \wedge \ a^{k+j} b^k \in L(M) \\ \Rightarrow & \qquad \langle\ j > 0 \Rightarrow k + j \neq k. \text{ Par conséquent, } a^{k+j} b^k \notin L.\ \rangle \\ & \exists M : \exists j, k \in \mathbf{N} : L(M) = L \ \wedge \ a^{k+j} b^k \in L(M) \ \wedge \ a^{k+j} b^k \notin L \\ \Rightarrow & \qquad \langle\ a^{k+j} b^k \in L(M) \ \wedge \ a^{k+j} b^k \notin L \ \Rightarrow \ L(M) \neq L\ \rangle \\ & \exists M : L(M) = L \ \wedge \ L(M) \neq L \\ \Leftrightarrow & \qquad \langle\ (p \wedge \neg p) \Leftrightarrow \mathsf{faux} \text{ et } (\exists x : \mathsf{faux}) \Leftrightarrow \mathsf{faux}.\ \rangle \end{array}$$

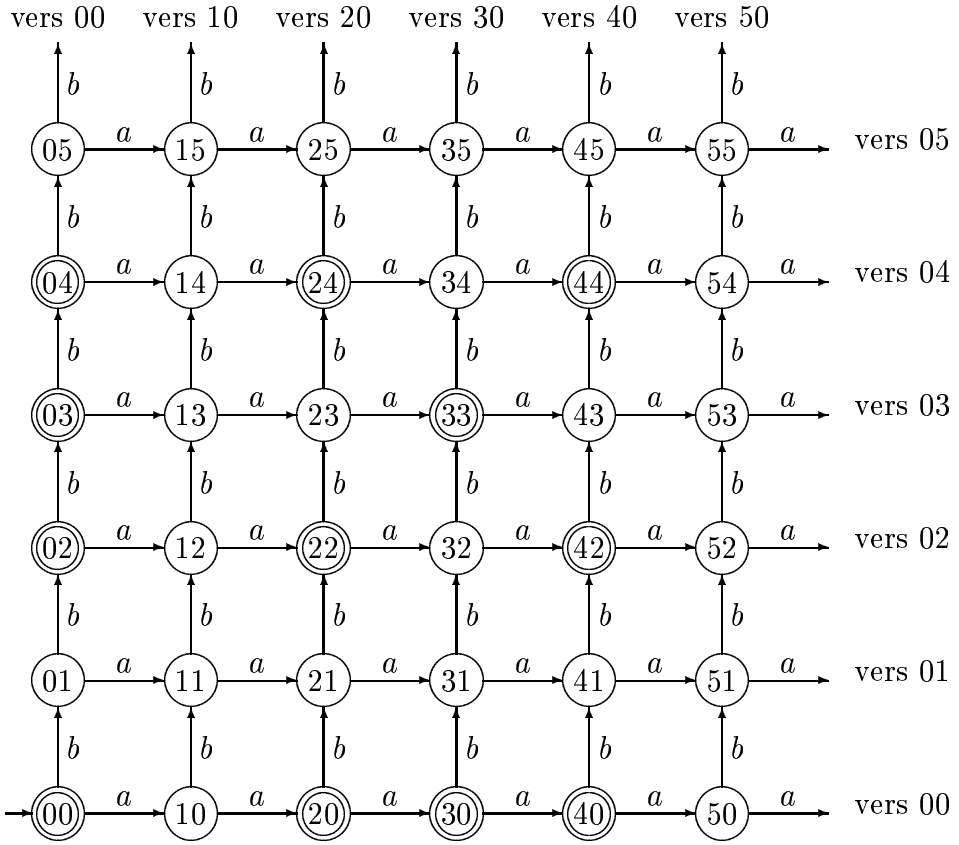
faux.

Problème 1, page 76. Montrez la régularité du langage sur $\{a, b\}$ dont les éléments sont exactement les séquences qui contiennent un nombre pair de a et un nombre pair de b ou un nombre de a divisible par 3 et un nombre de b divisible par 3.

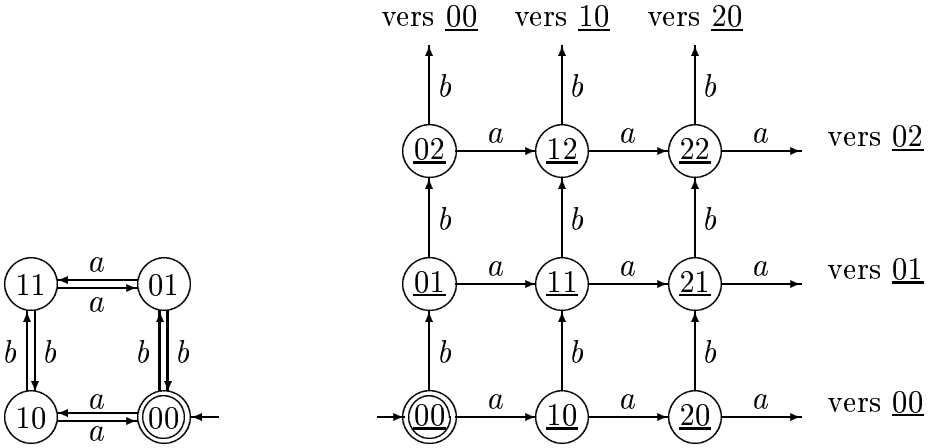
Solution. Pour montrer que L est régulier, construisons un automate fini déterministe qui le reconnaît. En tout temps lors de la lecture, l'automate doit savoir

- le reste de la division par 2 du nombre de a lus (2 cas) ;
- le reste de la division par 3 du nombre de a lus (3 cas) ;
- le reste de la division par 2 du nombre de b lus (2 cas) ;
- le reste de la division par 3 du nombre de b lus (3 cas).

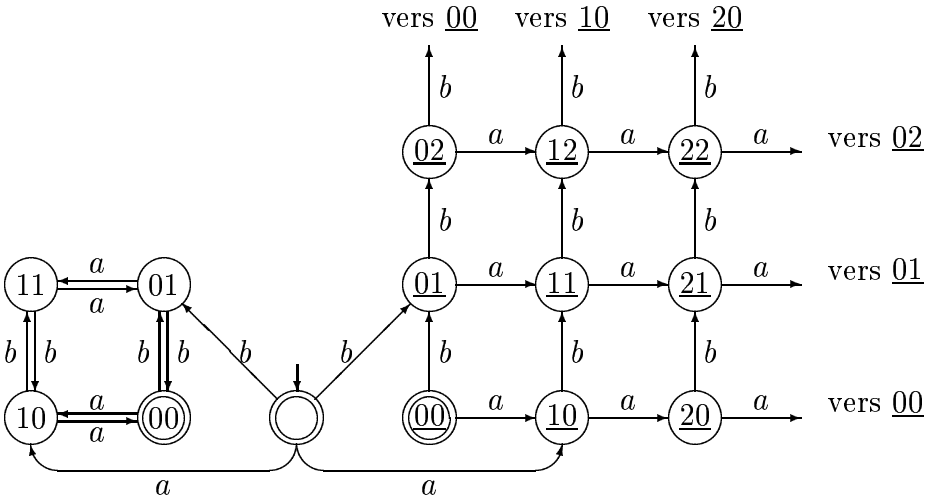
Il y a donc $2 \times 3 = 6$ cas pour les a et 6 cas pour les b, i.e. 36 cas différents en tout. L'automate doit avoir 36 états. Identifions chaque état par un nombre de la forme ij, où $i, j \in \{0, 1, 2, 3, 4, 5\}$; i et j indiquent le reste de la division par 6 du nombre de a et de b lus. L'état initial est 00. Notez que λ doit être acceptée ; l'état 00 doit donc aussi être final.



Cet automate consiste simplement en deux compteurs modulo 6, l'un pour les a, l'autre pour les b. Il est aussi possible de construire un automate fini non déterministe qui reconnaît le même langage. Voici tout d'abord deux automates finis déterministes ; le premier accepte les séquences contenant un nombre pair de a et un nombre pair de b et le second accepte les séquences contenant un nombre de a divisible par 3 et un nombre de b divisible par 3.



En utilisant la méthode du théorème 1.99, on fait l'union des automates. L'automate résultant de cette union est non déterministe et reconnaît le langage demandé.

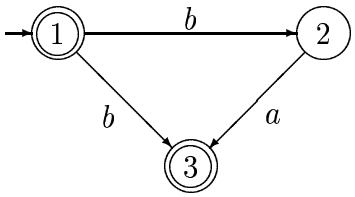


Problème 2, page 76. Montrez que chaque automate fini déterministe $(S, \Sigma, \delta, \iota, F)$ est aussi un automate fini non déterministe.

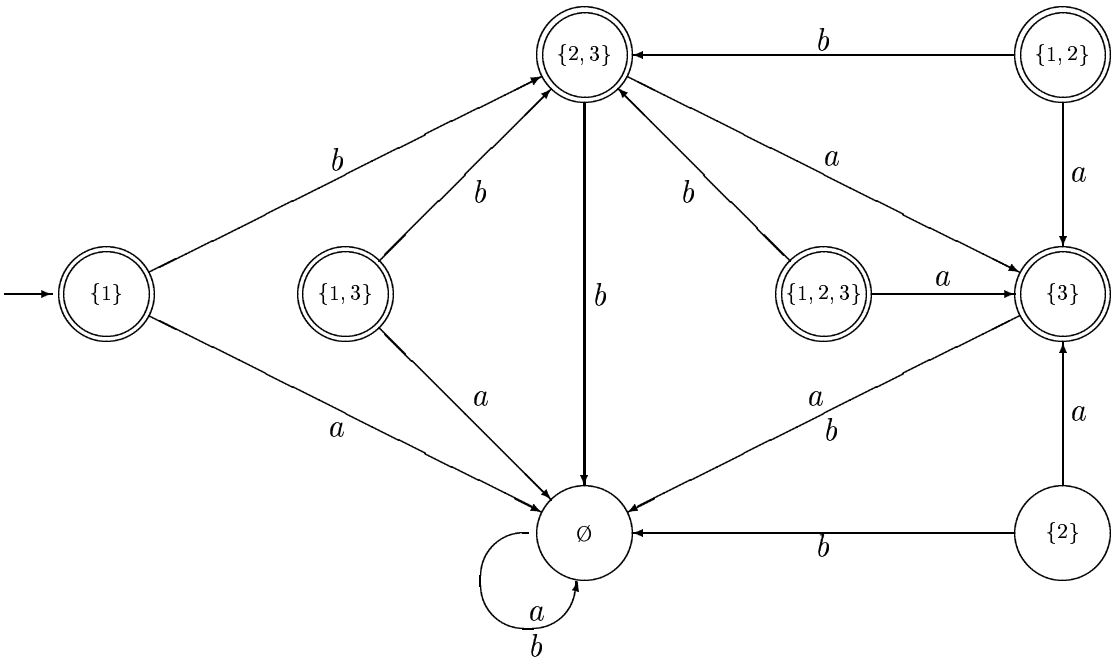
Solution. La seule composante qui distingue un automate fini non déterministe d'un automate fini déterministe, c'est la troisième. Dans le cas de l'automate non déterministe, c'est une relation. Dans le cas de l'automate déterministe, ça doit être une

fonction. Mais toute fonction est une relation (c'est un cas particulier de relation). Par conséquent, tout automate fini déterministe est aussi un automate fini non déterministe.

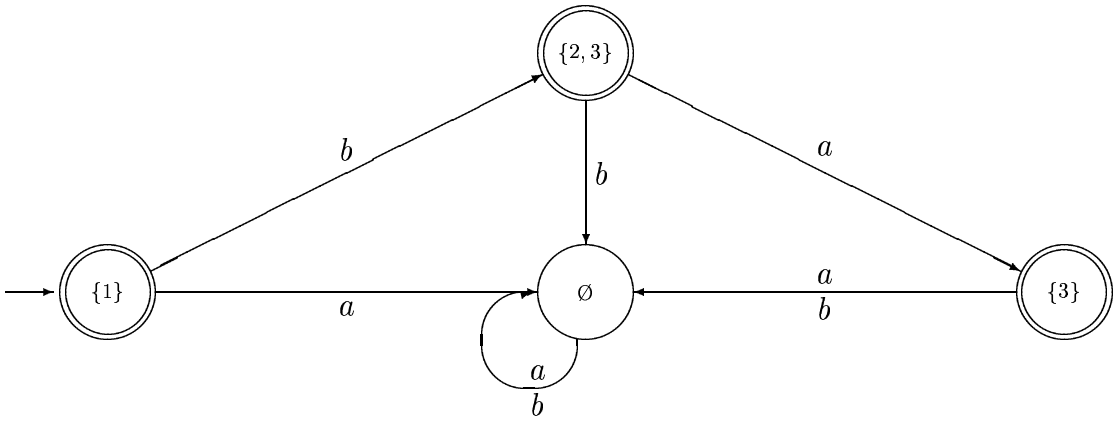
Problème 3, page 76. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un automate fini déterministe qui accepte les mêmes séquences que l'automate fini non déterministe ci-dessous. L'alphabet est $\{a, b\}$.



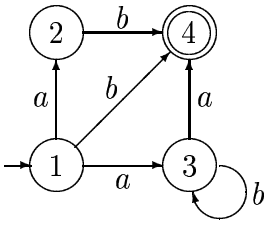
Solution. En utilisant les techniques du théorème 1.74, nous trouvons l'automate suivant.



Cet automate est une solution au problème. Il est possible de trouver une autre solution, plus simple, en supprimant les états qui ne peuvent être atteints à partir de l'état initial. L'automate suivant est le résultat d'une telle simplification.



Problème 4, page 76. En utilisant les techniques présentées dans la preuve du théorème 1.74, construisez un diagramme de transitions pour un automate fini déterministe qui accepte le même langage que l'automate fini non déterministe représenté ci-dessous. L'alphabet est $\{a, b\}$.



Solution. Appelons M cet automate. M a 4 états. Si l'on applique la recette du théorème 1.3, nous devrons dessiner un automate déterministe M' à 16 états (i.e. 2^4 états). Cela donne un diagramme plutôt encombré. Afin d'y voir clair, commençons par représenter les transitions au moyen d'une table. Nous dessinerons alors seulement les états qu'il est possible d'atteindre.

L'ensemble des états du nouvel automate est

$$S' = \{X : X \subseteq \{1, 2, 3, 4\}\} = \wp(\{1, 2, 3, 4\}).$$

L'état initial est $\{1\}$. L'ensemble des états finaux est

$$F' = \{X : X \in S' \text{ et } 4 \in X\},$$

car l'état 4 est le seul état final de l'automate original.

La table de transitions suivante est construite comme il est expliqué à la section 1.1.3. La manière la plus simple de la remplir est de commencer par énumérer les plus petits états de M' (i.e. ceux qui contiennent le moins grand nombre d'états de M). Il

est ensuite facile de calculer les autres en faisant l'union appropriée. Par exemple, si un a est lu dans l'état $\{2,3\}$, l'état de destination est obtenu en faisant l'union des états atteints à partir des états $\{2\}$ et $\{3\}$ à la lecture d'un a, c'est-à-dire $\emptyset$ et $\{4\}$, ce qui donne $\{4\}$.

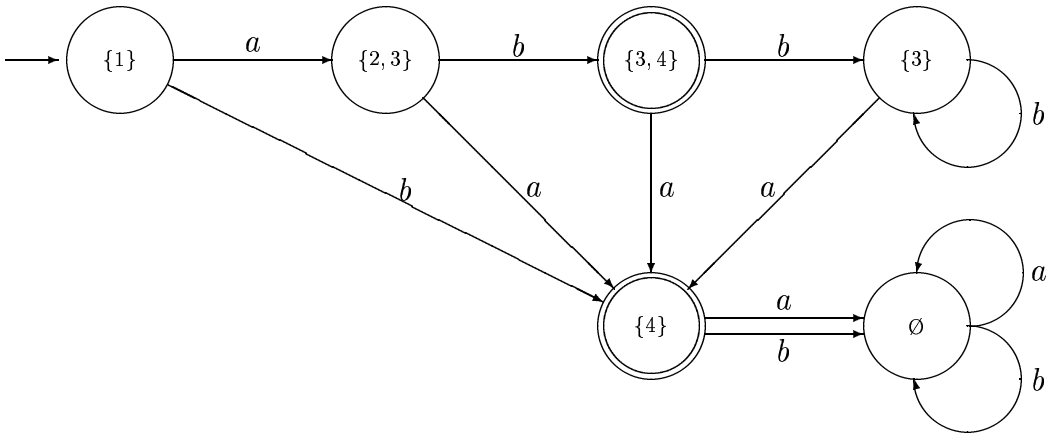
	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{1\}$	$\{2,3\}$	$\{4\}$
$\{2\}$	$\emptyset$	$\{4\}$
$\{3\}$	$\{4\}$	$\{3\}$
$\{4\}$	$\emptyset$	$\emptyset$
$\{1,2\}$	$\{2,3\}$	$\{4\}$
$\{1,3\}$	$\{2,3,4\}$	$\{3,4\}$
$\{1,4\}$	$\{2,3\}$	$\{4\}$

	a	b
$\{2,3\}$	$\{4\}$	$\{3,4\}$
$\{2,4\}$	$\emptyset$	$\{4\}$
$\{3,4\}$	$\{4\}$	$\{3\}$
$\{1,2,3\}$	$\{2,3,4\}$	$\{3,4\}$
$\{1,2,4\}$	$\{2,3\}$	$\{4\}$
$\{1,3,4\}$	$\{2,3,4\}$	$\{3,4\}$
$\{2,3,4\}$	$\{4\}$	$\{3,4\}$
$\{1,2,3,4\}$	$\{2,3,4\}$	$\{3,4\}$

L'état initial $\{1\}$ doit faire partie de l'automate déterministe. Ajoutons les états accessibles à partir de l'état $\{1\}$, i.e. $\{2,3\}$ et $\{4\}$, puis les états accessibles à partir de ces derniers, etc. Nous arrêtons cette procédure lorsqu'aucun état nouveau ne s'ajoute à la liste. Les états qui ne sont pas inclus ne sont pas accessibles et ne participent pas à la reconnaissance du langage. Remarquez que nous aurions aussi pu procéder de cette manière pour construire la table ou même l'automate directement (cette dernière option est celle qui demande le moins de travail). L'automate final est

$$M'' = (\{\emptyset, \{1\}, \{3\}, \{4\}, \{2,3\}, \{3,4\}\}, \{a,b\}, \{1\}, \delta, \{\{4\}, \{3,4\}\}),$$

où δ est la fonction de transition décrite par le diagramme suivant. Remarquez que M'' est totalement défini et non ambigu.



Problème 1, page 87. Construisez une grammaire régulière générant le langage

$$\{ab^m ab^n : m \in \mathbf{N} \wedge n \in \mathbf{N}\}.$$

Solution. Comme la génération des symboles se fait de gauche à droite avec une grammaire régulière, il est clair que la première règle doit avoir la forme $S \to aA$, où A est en charge de générer $b^m a b^n$. Ceci donne lieu à deux nouvelles règles, $A \to bA$ et $A \to aB$. Cette dernière règle termine la récursion qui génère la première série de b ; en particulier, elle s'applique tout de suite après la règle $S \to aA$ dans le cas où $m = 0$. Le symbole non terminal B est en charge de générer b^n. Les règles $B \to bB$ et $B \to \lambda$ font l'affaire. La grammaire est donc

$$S \to aA \qquad A \to bA \qquad A \to aB \qquad B \to bB \qquad B \to \lambda$$

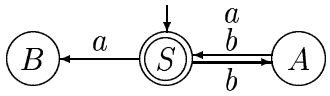
Problème 2, page 87. Donnez un automate fini qui reconnaît le même langage que le langage généré par la grammaire régulière suivante. Le symbole initial est S. Décrivez le langage dans vos propres mots.

$$\begin{array}{ll} S \to aB & A \to aS \\ S \to bA & A \to bS \\ S \to \lambda & \end{array}$$

Solution. Appliquons le théorème 1.94. Comme cette grammaire ne contient pas de règles de la forme *Non terminal* $\to$ *Terminal*, nous n'avons pas à les éliminer. L'automate correspondant à la grammaire ci-dessus est donc

$$M = (\{S, A, B\}, \{a, b\}, \rho, S, \{S\}),$$

où la relation ρ est donnée par le diagramme de transitions suivant.



Le langage généré par la grammaire (et reconnu par l'automate) est l'ensemble qui contient la séquence a ainsi que les séquences finies de a et de b de longueur paire qui ont un b dans les positions impaires (ceci inclut en particulier la séquence vide). On peut aussi l'exprimer de la manière suivante :

$$L = \{w : \text{pair}(|w|) \wedge (i \text{ impair } \Rightarrow w[i] = b)\}$$

(la notation $w[i]$ désigne le i^{e}élément de la séquence w). Au lieu de pair$(|w|)$, nous pourrions écrire $(\exists n \in \mathbf{N} : |w| = 2n)$.

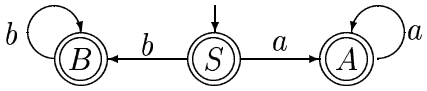
Problème 3, page 87. Dessinez le diagramme de transitions d'un automate fini qui accepte le langage généré par la grammaire régulière ci-dessous (le symbole de départ est S). Décrivez le langage dans vos propres mots.

$$\begin{array}{lll} S \to \lambda & B \to bB & A \to aA \\ S \to aA & B \to \lambda & A \to \lambda \\ S \to bB & & \end{array}$$

Solution. Appliquons le théorème 1.94. Comme cette grammaire ne contient pas de règles de la forme *Non terminal* $\to$ *Terminal*, nous n'avons pas à les éliminer. L'automate correspondant à la grammaire ci-dessus est donc

$$M = (\{S, A, B\}, \{a, b\}, \rho, S, \{S, A, B\}),$$

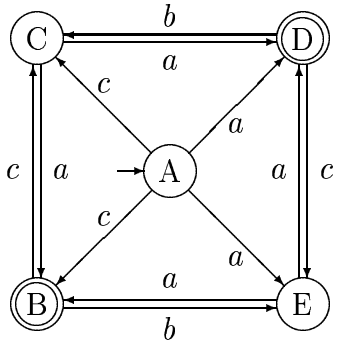
où la relation ρ est donnée par le diagramme de transitions suivant.



Le langage généré par la grammaire (et reconnu par l'automate) est l'ensemble des séquences finies de a et de b qui contiennent seulement des a ou seulement des b ou aucun des deux (i.e. λ). On peut aussi l'exprimer de la manière suivante :

$$L = \{a^n : n \in \mathbf{N}\} \cup \{b^n : n \in \mathbf{N}\} = \{a\}^* \cup \{b\}^*.$$

Problème 4, page 87. Donnez une grammaire régulière qui génère le même langage que le langage accepté par l'automate fini suivant.



Solution. Il suffit d'appliquer le théorème 1.94. Ceci conduit à la grammaire suivante :

$A \to cB$	$B \to cC$	$C \to aB$	$D \to bC$	$E \to aB$
$A \to cC$	$B \to bE$	$C \to aD$	$D \to cE$	$E \to aD$
$A \to aD$	$B \to \lambda$		$D \to \lambda$	
$A \to aE$				

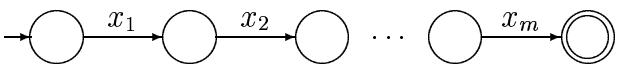
Problème 1, page 107. Montrez que tout langage fini est régulier.

Solution. Notons d'abord qu'un langage fini sur un alphabet Σ, c'est un sous-ensemble fini de Σ^*. Autrement dit, $L \subseteq \Sigma^*$ est un langage fini si sa cardinalité est un nombre naturel, i.e. $|L| \in \mathbf{N}$. Un langage fini contient donc une quantité finie de séquences ; le langage $\emptyset$ est un cas particulier de langage fini.

1. Méthode 1. Soit L un langage fini. Si $L = \emptyset$, alors L est régulier, puisque l'automate  l'accepte. Si $L \neq \emptyset$, alors

   $$L = L_1 \cup L_2 \cup \ldots \cup L_n = \bigcup_{i=1}^{n} L_i,$$

   où chaque L_i est un singleton et $n \in \mathbf{N}^+$. Il est évident que chaque L_i est un langage régulier, puisque $L_i = \{x_1 x_2 \ldots x_m\}$ est reconnu par l'automate suivant.

   (Remarquez que si $m = 0$, alors $L_i = \{x_1 x_2 \ldots x_m\} = \{\lambda\}$ est accepté par l'automate  . C'est un cas limite.)

   Il ne reste plus qu'à voir qu'une union finie de langages réguliers est un langage régulier. Procédons par induction sur n. Soit $P(n)$ le prédicat

   $$\bigcup_{i=1}^{n} L_i \text{ est un langage régulier}$$

   Il faut montrer que $P(n)$ est vrai pour tout $n \in \mathbf{N}^+$.
   - Base d'induction : $\bigcup_{i=1}^{1} L_i = L_1$ est régulier et donc $P(1)$ est vrai.
   - Étape d'induction : Il faut montrer que $P(k) \Rightarrow P(k+1)$.

     $P(k)$
     $\Rightarrow$ $\langle$ Définition de P. $\rangle$
     $\bigcup_{i=1}^{k} L_i$ est régulier
     $\Rightarrow$ $\langle$ L'union de deux langages réguliers est un langage régulier (théorème 1.99). $\rangle$

$(\bigcup_{i=1}^{k} L_i) \cup L_{k+1}$ est régulier

$\Rightarrow$ $\langle$ Définition de la notation $\bigcup_{i=m}^{n} x_i$. $\rangle$

$\bigcup_{i=1}^{k+1} L_i$ est régulier

$\Rightarrow$ $\langle$ Définition de P. $\rangle$

$P(k+1)$.

2. Méthode 2. La preuve suivante utilise les grammaires régulières (on peut aussi faire une preuve à partir des expressions régulières). Soient $w_1, w_2, \ldots, w_n$ les n séquences de L. Il est évident que la grammaire (V, T, S, R) avec

$$R = \{S \to w_1, S \to w_2, \ldots, S \to w_n\}$$

génère L (notez en particulier que le nombre de règles est fini, ce qui est exigé par la définition de grammaire). Remarquez aussi que la grammaire $(\{S\}, \{a\}, S, \emptyset)$ permet de générer le langage $\emptyset$.

Cependant cette grammaire n'est pas régulière. Le problème vient des règles de la forme

$$S \to x_1 x_2 \ldots x_k \text{ (où } x_i \in T, \text{ pour } 1 \leq i \leq k)$$

lorsque $k \geq 2$ (ces règles génèrent les séquences de 2 symboles ou plus). Ces règles ne sont pas admises dans une grammaire régulière. Il faut donc remplacer chacune des règles de la forme ci-dessus par l'ensemble des règles suivantes :

$$\begin{aligned} S &\to x_1 X_1 \\ X_1 &\to x_2 X_2 \\ &\vdots \\ X_{k-2} &\to x_{k-1} X_{k-1} \\ X_{k-1} &\to x_k \end{aligned}$$

Dans ces règles, chaque X_i est un nouveau symbole non terminal.

Comme le nombre de séquences du langage est fini et que le nombre de symboles de chaque séquence est fini, le nombre de règles est fini. De plus, chaque règle se conforme aux restrictions imposées aux grammaires régulières. La grammaire finale est donc régulière.

Exemple. Soit $L = \{\lambda, a, ba, aba\}$. La grammaire suivante est régulière et génère L :

$$(\{S, X_1, X_2, X_3\}, \{a, b\}, S, R)$$

où R est l'ensemble des règles

$S \to \lambda$	$S \to bX_1$	$X_1 \to a$	$X_3 \to a$
$S \to a$	$S \to aX_2$	$X_2 \to bX_3$	

Bien sûr il est possible de diminuer le nombre de symboles non terminaux introduits. Ainsi, la grammaire qui suit génère le même langage que celle qui

précède. Toutefois, il n'est pas nécessaire de tenir compte de cette possibilité de simplification dans la démonstration ci-dessus.

$$G = (\{S, X_1, X_2\}, \{a, b\}, S, R)$$

où R est l'ensemble des règles

$S \to \lambda$	$S \to bX_1$	$X_1 \to a$
$S \to a$	$S \to aX_2$	$X_2 \to bX_1$

Problème 2, page 107. Décrivez (par des mots ou par une expression ensembliste) le langage représenté par chacune des expressions régulières suivantes.

(a) $((c \cup b)^* \circ a)$
(b) $((a \circ a^*) \circ (b \circ b^*))$
(c) $((a \circ a^*) \cup (b \circ b^*))$
(d) $((a^* \circ b^*) \circ c^*)$

Solution.

(a) Une séquence du langage consiste en une suite de 0 ou plusieurs c et/ou b dans un ordre quelconque, terminée par un a :

$$L(((c \cup b)^* \circ a)) = \{b, c\}^* \circ \{a\}$$

(b) Une séquence du langage est formée de 1 ou plusieurs a suivis de 1 ou plusieurs b :

$$L(((a \circ a^*) \circ (b \circ b^*))) = \{a^m b^n : m, n \in \mathbf{N}^+\}.$$

(c) Une séquence de symboles du langage est soit une suite de 1 ou plusieurs a, soit une suite de 1 ou plusieurs b :

$$\begin{aligned} & L(((a \circ a^*) \cup (b \circ b^*))) \\ = \ & \{a^n : n \in \mathbf{N}^+\} \cup \{b^n : n \in \mathbf{N}^+\} \\ = \ & \{s^n : (s = a \vee s = b) \wedge n \in \mathbf{N}^+\}. \end{aligned}$$

(d) Une séquence du langage consiste en 0 ou plusieurs a, suivis de 0 ou plusieurs b, suivis de 0 ou plusieurs c :

$$L(((a^* \circ b^*) \circ c^*)) = \{a^l b^m c^n : l, m, n \in \mathbf{N}\}.$$

Problème 3, page 107. Trouvez un diagramme de transitions qui accepte le langage généré par la grammaire régulière suivante. (Le symbole de départ est S.) Ensuite, trouvez une expression régulière représentant le même langage.

$S \to aN$	$M \to cN$	$N \to bM$
$S \to a$	$M \to c$	$N \to b$

Solution.

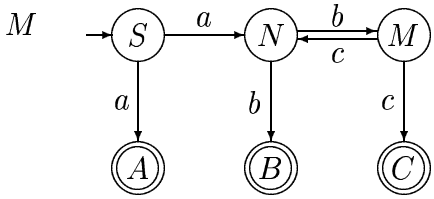
1. Solution 1. Appliquons la méthode du théorème 1.94. Appelons G la grammaire du problème. Il faut d'abord transformer G en une grammaire qui ne contient pas de productions de la forme

   $$Nonterminal \to Terminal.$$

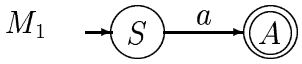
   Une telle grammaire est la grammaire $G' = (\{S, M, N, A, B, C\}, \{a, b, c\}, S, R)$ où R est l'ensemble de règles de réécriture suivant :

$S \to aN$	$M \to cN$	$N \to bM$	$A \to \lambda$
$S \to aA$	$M \to cC$	$N \to bB$	$B \to \lambda$
			$C \to \lambda$

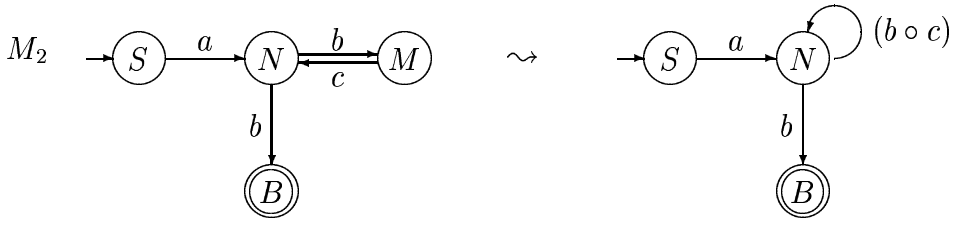
   (au lieu d'introduire trois nouveaux symboles, nous aurions aussi pu en introduire un seul). Le diagramme de transitions qui accepte le langage généré par G' est

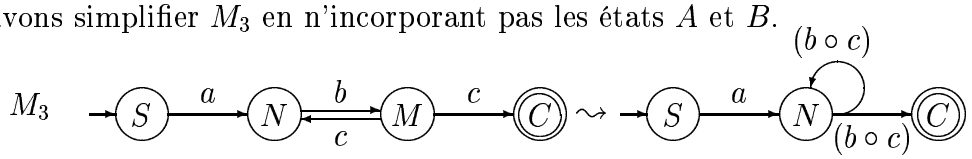
   En utilisant la méthode du théorème 1.122, cherchons l'expression régulière représentant le même langage. Comme la machine M a trois états finaux, nous devons la séparer en trois machines ayant chacune un seul état final. La machine M_1 est comme M, mais avec A comme seul état final. Comme les séquences qui conduisent aux états autres que S et A ne peuvent être acceptées, nous pouvons simplifier M_1 en

   L'expression régulière er_1 telle que $L(er_1) = L(M_1)$ est $er_1 = a$.
   La machine M_2 est comme M, mais avec B comme seul état final. Comme les séquences qui conduisent aux états A et C ne peuvent être acceptées, nous pouvons simplifier M_2 en n'incorporant pas les états A et C.

L'expression régulière er_2 telle que $L(er_2) = L(M_2)$ est $er_2 = ((a \circ (b \circ c)^*) \circ b)$. La machine M_3 est comme M, mais avec C comme seul état final. Comme les séquences qui conduisent aux états A et B ne peuvent être acceptées, nous pouvons simplifier M_3 en n'incorporant pas les états A et B.



L'expression régulière er_3 telle que $L(er_3) = L(M_3)$ est $er_3 = ((a \circ (b \circ c)^*) \circ (b \circ c))$. L'expression régulière qui représente le langage accepté par l'automate M est donc

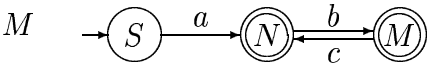
$$\begin{aligned} & er \\ = \ & ((er_1 \cup er_2) \cup er_3) \\ = \ & ((a \cup ((a \circ (b \circ c)^*) \circ b)) \cup ((a \circ (b \circ c)^*) \circ (b \circ c))). \end{aligned}$$

2. Solution 2. Examinons plus attentivement la grammaire G (la grammaire du problème). Remplaçons-y la règle $N \to b$ par la règle $M \to \lambda$. Cela ne change pas le langage généré, car
   - l'effet de la règle $N \to b$ dans G est obtenu par la dérivation $N \Rightarrow bM \Rightarrow b$ dans la nouvelle grammaire.
   - l'ajout de la règle $M \to \lambda$ ne permet pas de dériver de nouvelles chaînes. Cela pourrait se produire, par exemple, si G contenait une règle comme $V \to aM$ et pas de règle $V \to a$.

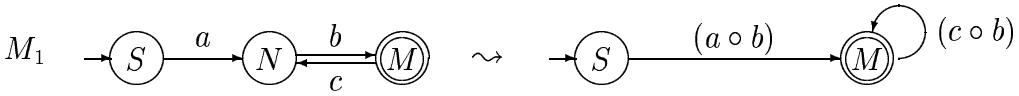
   De même, on peut remplacer les règles $S \to a$ et $M \to c$ par la règle $N \to \lambda$. On obtient ainsi la grammaire régulière $G' = (\{S, M, N\}, \{a, b, c\}, S, R)$ dont les règles sont

$$\begin{array}{lll} S \to aN & M \to cN & N \to bM \\ & M \to \lambda & N \to \lambda \end{array}$$

   En utilisant les mêmes méthodes que pour la solution 1, nous trouvons que l'automate M correspondant à G' est



   Cherchons l'expression régulière représentant $L(M)$. Comme la machine M a deux états finaux, nous devons la séparer en deux machines ayant chacune un seul état final. La machine M_1 est comme la machine M, mais avec M comme seul état final (que de M dans le décor).



L'expression régulière er_1 telle que $L(er_1) = L(M_1)$ est $er_1 = ((a \circ b) \circ (c \circ b)^*)$. La machine M_2 est comme la machine M, mais avec N comme seul état final.

L'expression régulière er_2 telle que $L(er_2) = L(M_2)$ est $er_2 = (a \circ (b \circ c)^*)$. L'expression régulière qui représente le langage accepté par l'automate M est donc

$$er = (er_1 \cup er_2) = (((a \circ b) \circ (c \circ b)^*) \cup (a \circ (b \circ c)^*)).$$

Problème 4, page 107. Trouvez une expression régulière qui représente l'intersection des langages représentés par chacune des paires d'expressions régulières qui suivent.

(a) $(a \cup b^*)$ et $(a \cup b)^*$

(b) $(a \circ (a \cup b)^*)$ et $((a \cup b)^* \circ b)$

(c) $(((a \cup b) \circ b) \circ (a \cup b)^*)$ et $(b \circ (a \cup b)^*)$

(d) $((a \circ b)^* \circ (a \cup \emptyset))$ et $((a \cup b) \circ (a \circ b)^*)$

Solution. Rappelons d'abord comment on définit le langage représenté par une expression régulière sur un alphabet Σ (définition 1.117) :

$$\begin{array}{lll} L(\emptyset) = \emptyset & L((p \cup q)) = L(p) \cup L(q) & L(p^*) = (L(p))^* \\ L(x) = \{x\} & L((p \circ q)) = L(p) \circ L(q) & \end{array}$$

où $x \in \Sigma$ et où p et q sont des expressions régulières.

(a) $L((a \cup b)^*) = (L((a \cup b)))^* = (L(a) \cup L(b))^* = (\{a\} \cup \{b\})^* = \{a, b\}^*$. Le langage représenté par $(a \cup b)^*$ est l'ensemble des séquences finies de a et de b. Le langage $L((a \cup b^*))$ est donc inclus dans $L((a \cup b)^*)$. Par conséquent,

$$L((a \cup b^*)) \cap L((a \cup b)^*) = L((a \cup b^*)).$$

L'expression régulière recherchée est donc $(a \cup b^*)$.

(b) En utilisant le résultat $L((a \cup b)^*) = \{a, b\}^*$, trouvé au numéro précédent, nous avons

$$L((a \circ (a \cup b)^*)) = L(a) \circ L((a \cup b)^*) = \{a\} \circ \{a, b\}^*,$$
$$L((a \cup b)^* \circ b) = L((a \cup b)^*) \circ L(b) = \{a, b\}^* \circ \{b\}.$$

Le premier langage est l'ensemble des séquences finies de a et de b qui débutent par un a ; le deuxième langage est l'ensemble des séquences finies de a et de b qui

se terminent par un b. L'intersection de ces deux langages est donc l'ensemble des séquences finies de a et de b qui commencent par un a et se terminent par un b, i.e.

$$\{a\} \circ \{a, b\}^* \circ \{b\}.$$

Remarquez que nous n'avons pas besoin de parenthèses dans l'expression ensembliste ci-dessus. En effet, la concaténation de langages est une opération associative. En appliquant les règles données ci-dessus, il est facile de vérifier que l'expression régulière $((a \circ (a \cup b)^*) \circ b)$ représente ce langage, ainsi que l'expression $(a \circ ((a \cup b)^* \circ b))$.

(c) Maintenant que nous avons appliqué les règles quelques fois, il est facile de trouver directement le langage représenté par les expressions régulières données :

$$L((((a \cup b) \circ b) \circ (a \cup b)^*)) = \{a, b\} \circ \{b\} \circ \{a, b\}^*,$$
$$L((b \circ (a \cup b)^*)) = \{b\} \circ \{a, b\}^*.$$

Le premier langage est l'ensemble des séquences finies de a et de b qui contiennent au moins deux symboles et dont le deuxième symbole est un b. Le deuxième langage est l'ensemble des séquences finies de a et de b qui débutent par un b. L'intersection de ces deux langages est donc l'ensemble des séquences finies de a et de b dont les deux premiers symboles sont des b, i.e.

$$\{bb\} \circ \{a, b\}^*.$$

Deux expressions régulières représentant ce langage sont $((b \circ b) \circ (a \cup b)^*)$ et $(b \circ (b \circ (a \cup b)^*))$.

(d) A cause de la présence du symbole $\emptyset$ dans la première expression, faisons les choses au long pour voir ce qu'il lui arrive.

$$\begin{aligned}
& L(((a \circ b)^* \circ (a \cup \emptyset))) \\
= \; & L((a \circ b)^*) \circ L((a \cup \emptyset)) \\
= \; & (L((a \circ b)))^* \circ (L(a) \cup L(\emptyset)) \\
= \; & (L(a) \circ L(b))^* \circ (\{a\} \cup \emptyset) \\
= \; & (\{a\} \circ \{b\})^* \circ \{a\} \\
= \; & \{ab\}^* \circ \{a\}.
\end{aligned}$$

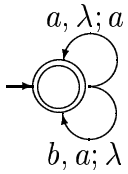
De même, nous trouvons

$$\begin{aligned}
& L(((a \cup b) \circ (a \circ b)^*)) \\
= \; & L((a \cup b)) \circ L((a \circ b)^*) \\
= \; & (L(a) \cup L(b)) \circ (L((a \circ b)))^* \\
= \; & (\{a\} \cup \{b\}) \circ (L(a) \circ L(b))^* \\
= \; & \{a, b\} \circ (\{a\} \circ \{b\})^* \\
= \; & \{a, b\} \circ \{ab\}^*.
\end{aligned}$$

Le premier langage est l'ensemble des séquences finies de a et de b qui sont formées de a et de b alternés, et qui commencent et se terminent par un a (exemples : $a, ababa$). Le deuxième langage est l'ensemble des séquences finies de a et de b qui débutent soit par un a, soit par un b, et qui se poursuivent avec 0 ou plusieurs répétitions de la sous-séquence ab (exemples : $a, b, aab, babab$). La seule séquence du deuxième langage qui commence et finit par un a est a. Cette séquence appartient aussi au premier langage. L'intersection des deux langages est donc le langage $\{a\}$. Ce langage est représenté par l'expression régulière a.

5.3 Chapitre 2

Problème 1, page 119. Quel est le langage accepté par l'automate à pile dont le diagramme de transitions est le suivant ?



Solution. L'automate est toujours dans un état final et accepte donc la séquence vide λ. Par la transition du haut, un a est empilé pour chaque a lu. Par la transition du bas, un a est dépilé pour chaque b lu. Une séquence est donc acceptée si et seulement si, à tout instant lors de la lecture, le nombre de b lus n'excède pas le nombre de a lus. De plus, une chaîne contenant un symbole autre que a ou b est rejetée.

Voici maintenant une autre manière, plus rigoureuse, de définir le langage accepté. Supposons, pour simplifier, que $\Sigma = \{a, b\}$. Dénotons par $w[1]$ le premier élément d'une séquence non vide. Définissons une fonction $OK : \Sigma^* \to \{vrai, faux\}$:

$$OK(w) = \begin{cases} vrai & \text{si } w \in \{a\}^* \qquad (\text{note : inclut } w = \lambda) \\ faux & \text{si } w[1] = b \\ OK(w_1 w_2) & \text{si } w_1 \in \{a\}^*,\ w_2 \in \Sigma\text{"}* \text{ et } w = w_1 a b w_2 \end{cases}$$

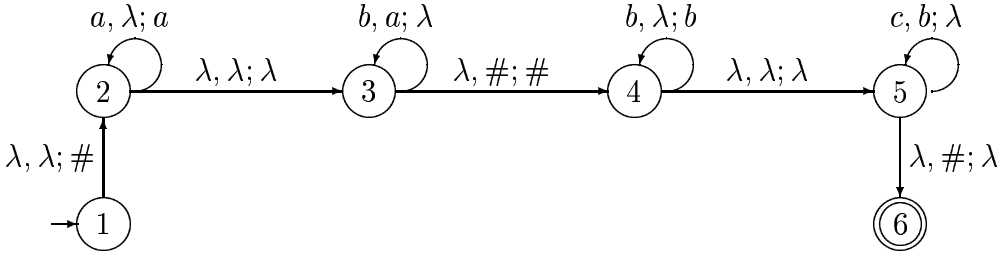
Il est facile de voir que si $w \notin \{a\}^*$ et si $w[1] \neq b$, alors w doit contenir une sous-séquence ab. En enlevant cette sous-séquence de w (troisième cas), on obtient une séquence qui appartient au langage si et seulement si w elle-même appartient au langage.

Le langage accepté par l'automate est alors $\{w \in \Sigma^* : OK(w)\}$.

Problème 2, page 119. Construisez un automate à pile qui reconnaît le langage

$$\{a^r b^s c^t : r \in \mathbf{N} \wedge s \in \mathbf{N} \wedge t \in \mathbf{N} \wedge s = r + t\}.$$

Solution. La stratégie consiste à empiler les a lus, puis à dépiler un a pour chaque b lu. Lorsque la pile est vide, le nombre de a lus égale le nombre de b lus. Il suffit alors de continuer à lire les b, en les empilant, puis de dépiler un b pour chaque c lu. La pile doit être vide après avoir lu le dernier symbole de la chaîne d'entrée. Il faut faire attention au fait que λ ainsi que les chaînes de la forme $a^r b^r$ ou $b^t c^t$ doivent être acceptées. Un automate non déterministe acceptant le langage donné est le suivant.



Problème 1, page 136. Construisez une grammaire non contextuelle qui génère le langage

$$\{a^r b^s c^t : r \in \mathbf{N} \wedge s \in \mathbf{N} \wedge t \in \mathbf{N} \wedge s = r + t\}.$$

Solution. On remarque que le langage à générer peut s'exprimer comme la concaténation de deux langages que nous connaissons bien :

$$\begin{aligned} & \{a^r b^s c^t : r, s, t \in \mathbf{N} \wedge s = r + t\} \\ = \ & \{a^r b^r b^t c^t : r, t \in \mathbf{N}\} \\ = \ & \{a^n b^n : n \in \mathbf{N}\} \circ \{b^n c^n : n \in \mathbf{N}\}. \end{aligned}$$

Nous connaissons une grammaire qui génère $\{a^n b^n : n \in \mathbf{N}\}$ (voir exercice 2.28). Il est alors facile de l'adapter à notre langage.

$$\begin{array}{lll} S \to GD & G \to aGb & D \to bDc \\ & G \to \lambda & D \to \lambda \end{array}$$

On voit que les dérivations à partir de G introduisent toujours un a avec un b alors que celles qui partent de D introduisent toujours un b avec un c. Parce que G est à gauche de D, les a précèdent les b qui précèdent les c.

Problème 2, page 136. Convertissez la grammaire suivante, dont le symbole initial est S, en grammaire sous forme normale de Chomsky.

$$\begin{array}{lll} S \to McN & M \to aM & N \to bN \\ & M \to \lambda & N \to \lambda \end{array}$$

Solution. Appelons G la grammaire donnée ci-dessus. On remarque d'abord que $\lambda \notin L(G)$, car toute séquence de $L(G)$ contient au moins un c, d'après la première règle. Il est donc possible de convertir G sous la forme normale de Chomsky.

1. Éliminons d'abord les règles-λ selon la méthode du théorème 2.32.
   (a) Calcul de $\mathcal{U}$. Il n'y a que deux chaînes-λ, soit $M \to \lambda$ et $N \to \lambda$, et elles sont toutes deux de longueur 0. On a donc $\mathcal{U} = \mathcal{U}_0 = \{M, N\}$.
   (b) Ajout de règles par élimination des symboles de $\mathcal{U}$ dans les règles de G. On doit ajouter les règles
   $$S \to cN \qquad S \to Mc \qquad S \to c \qquad M \to a \qquad N \to b$$
   (c) Retrait des règles-λ. Les règles restantes sont :
   $$\begin{array}{llll} S \to McN & S \to Mc & M \to aM & N \to bN \\ S \to cN & S \to c & M \to a & N \to b \end{array}$$
2. Application du théorème 2.36.
   (a) Remplacer chaque symbole terminal x par un nouveau symbole non terminal X et ajouter la règle $X \to x$.
   $$\begin{array}{lll} S \to MCN & M \to AM & A \to a \\ S \to CN & M \to A & B \to b \\ S \to MC & N \to BN & C \to c \\ S \to C & N \to B & \end{array}$$
   (b) Transformer le côté droit des règles ayant plus de 2 symboles non terminaux.
   $$\begin{array}{lll} S \to MR & M \to AM & A \to a \\ S \to CN & M \to A & B \to b \\ S \to MC & N \to BN & C \to c \\ S \to C & N \to B & R \to CN \end{array}$$
   (c) Élimination des règles ayant un seul symbole non terminal du côté droit. Il n'y a que trois chaînes impliquant de telles règles. Ces chaînes sont $S \to C$, $M \to A$ et $N \to B$. A cause des règles $A \to a$, $B \to b$ et $C \to c$, il suffit de remplacer $S \to C$ par $S \to c$, $M \to A$ par $M \to a$ et $N \to B$ par $N \to b$.
   $$\begin{array}{lll} S \to MR & M \to AM & A \to a \\ S \to CN & M \to a & B \to b \\ S \to MC & N \to BN & C \to c \\ S \to c & N \to b & R \to CN \end{array}$$

L'intérêt du théorème 2.36 est qu'il montre que la transformation peut toujours se faire (si $\lambda \notin L(G)$). Cependant la grammaire résultante est relativement complexe. Il est possible de trouver une grammaire plus simple, sous forme normale de Chomsky, générant le même langage que G. La grammaire suivante est celle qui a été proposée par Louis Choinière et Frédéric Guérin (1990).

$$S \to AS \qquad S \to SB \qquad S \to c \qquad A \to a \qquad B \to b$$

Problème 3, page 136. Soit $\Sigma = \{a, b, \circ, \cup, *,), (, \emptyset\}$. Construisez un automate à pile M tel que $L(M)$ est l'ensemble de toutes les séquences de Σ^* qui sont des expressions régulières sur l'alphabet $\{a, b\}$. Suggestion : Il est plus facile de trouver une grammaire non contextuelle qui génère le langage demandé que de construire l'automate ; une approche simple consiste donc à trouver la dite grammaire, puis à appliquer le théorème 2.26.

Solution. Il faut construire un automate à pile reconnaissant la syntaxe des expressions régulières. Cette syntaxe est décrite à la définition 1.115, page 97.

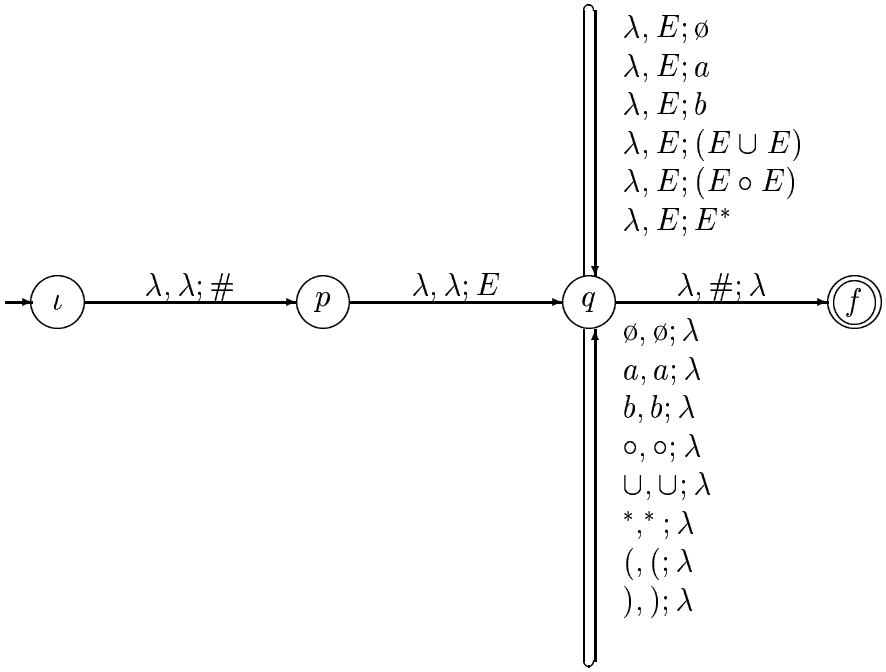
1. Solution 1. La solution la plus simple consiste à trouver la grammaire générant le langage des expressions régulières sur l'alphabet $\{a, b\}$. À partir de la syntaxe donnée à la définition 1.115, on trouve directement la grammaire

   $$G = (\{E\}, \{a, b, \circ, \cup, ^*, (,), \emptyset\}, E, R),$$

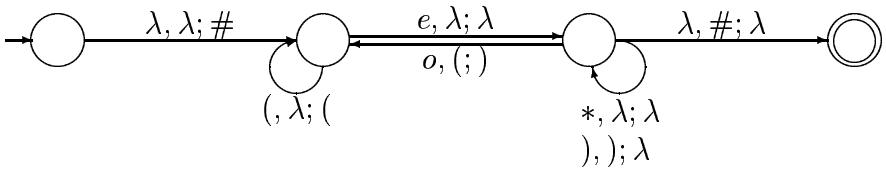
   où R est l'ensemble des règles qui suivent.

   $$\begin{array}{ll} E \to \emptyset & E \to (E \cup E) \\ E \to a & E \to (E \circ E) \\ E \to b & E \to E^* \end{array}$$

   En utilisant le théorème 2.26, on trouve alors l'automate à pile non déterministe suivant.



2. Solution 2. Il est plus difficile de trouver directement (sans passer par la grammaire) un automate qui fait le même travail. L'automate suivant est le résultat de modifications mineures aux solutions proposées par Roméo Guérin, David Renaud et Claude St-Pierre (1990).
   L'alphabet d'entrée de cet automate est le Σ du problème. Son alphabet de pile est $\Gamma = \{\#, (,)\}$. Les symboles e et o sont utilisés dans le diagramme pour éviter d'encombrer celui-ci inutilement ; il suffit de remplacer e par l'un des symboles de l'ensemble $\{\emptyset, a, b\}$ (symboles élémentaires) et o par l'un des symboles de l'ensemble $\{\circ, \cup\}$ (opérateurs binaires).



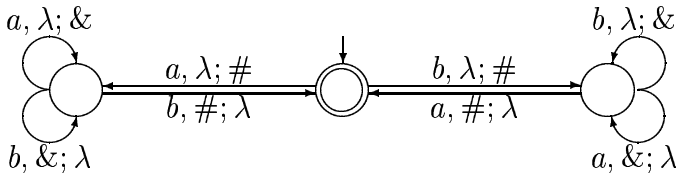
Problème 4, page 136. Donnez une grammaire non contextuelle qui génère le langage sur l'alphabet $\{a, b\}$ dont les éléments sont les séquences qui contiennent le même nombre de a que de b.

Solution. La grammaire non contextuelle suivante génère L. Pour s'en convaincre, remarquons que λ fait partie du langage généré et que les chaînes non vides du langage peuvent toutes se décomposer en sous-chaînes de la forme $avbw$ ou $bvaw$, où v et w sont des sous-séquences ayant un nombre égal de a et de b.

$S \to aSbS$ $\qquad$ $S \to bSaS$ $\qquad$ $S \to \lambda$

Problème 1, page 155. Construisez un automate à pile déterministe qui reconnaît le langage sur l'alphabet $\{a, b\}$ dont les éléments sont les séquences qui contiennent le même nombre de a que de b.

Solution. La partie à gauche de l'état initial reconnaît les sous-séquences commençant par un a et ayant un nombre égal de a et de b. La partie à droite de l'état initial reconnaît les sous-séquences commençant par un b et ayant un nombre égal de a et de b. Remarquez que l'alphabet de pile est $\Gamma = \{\&, \#\}$ et n'a que deux symboles, ce qui fait qu'il y a peu de combinaisons symbole lu/symbole dépilé. Il est par conséquent facile de vérifier que l'automate est totalement défini.



Problème 2, page 155. Construisez un automate à pile déterministe M tel que $L(M)$ est le langage généré par la grammaire ci-dessous. Le symbole initial est S.

$$S \to abN \qquad N \to cS \qquad N \to c \qquad N \to \lambda$$

Solution. La manière la plus simple de procéder semble être de transformer légèrement la grammaire afin de déterminer plus facilement le langage généré par celle-ci. Il est ensuite facile de construire l'automate demandé.

En remplaçant le symbole N dans la règle $S \to abN$ par le côté droit des règles où N apparaît à gauche, on obtient la grammaire suivante, qui est équivalente à la grammaire originale.

$$\begin{array}{ll} S \to abcS & N \to cS \\ S \to abc & N \to c \\ S \to ab & N \to \lambda \end{array}$$

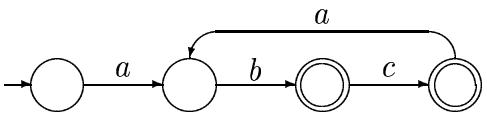
On remarque maintenant que les règles de droite ne seront jamais utilisées. On peut donc les supprimer. Cela donne la grammaire

$$S \to abcS \qquad S \to abc \qquad S \to ab$$

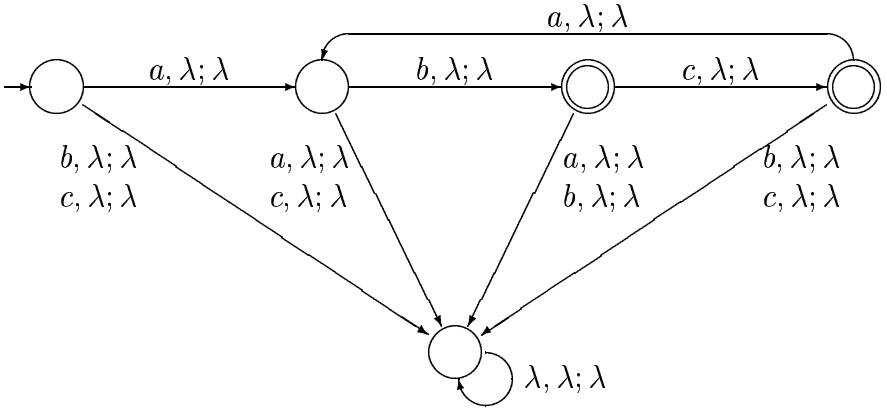
Le langage généré par cette grammaire est

$$\{(abc)^n abc : n \in \mathbf{N}\} \cup \{(abc)^n ab : n \in \mathbf{N}\}.$$

Ce langage est reconnu par l'automate fini suivant.



Remarquez que cet automate n'est pas ambigu, mais n'est pas totalement défini. Il faut donc ajouter un état poubelle. De plus, on nous demande un automate à pile. Puisque le travail peut être fait par un automate fini, l'automate à pile requis n'a pas à utiliser sa pile. Voici l'automate final.



Au lieu de trouver le langage généré par la grammaire, puis l'automate reconnaissant le langage, nous pouvons aussi procéder directement à partir de la grammaire, en transformant celle-ci. On remplace d'abord la règle $S \to abN$ par les règles $S \to aM$ et $M \to bN$; ensuite on remplace la règle $N \to cS$ par les règles $N \to cC$ et $C \to aM$; finalement, on remplace la règle $N \to c$ par la règle $C \to \lambda$. Cela nous donne la grammaire régulière suivante, qui est équivalente à la grammaire originale.

$S \to aM$	$N \to cC$	$C \to aM$
$M \to bN$	$N \to \lambda$	$C \to \lambda$

En utilisant le théorème 1.94, on trouve directement le premier automate ci-dessus.

Problème 3, page 155. Utilisez le corollaire 2.42 pour montrer que le langage

$$\{a^p : p \in \mathbf{N} \wedge p \text{ est premier}\}$$

n'est pas un langage non contextuel.

Solution. Servons-nous du corollaire 2.42, puisqu'il est équivalent au théorème 2.40 et qu'il est plus facile à utiliser que celui-ci. Comme le langage $\{a^p : p \in \mathbf{N} \wedge p \text{ est premier}\}$ est infini, nous pouvons appliquer le corollaire.

Considérons une séquence quelconque $a^p \in L$ et supposons qu'elle est décomposée sous la forme $svuwt$, où au moins l'une des sous-séquences v ou w est non vide. Il faut montrer qu'alors il existe un $n \in \mathbf{N}^+$ tel que $sv^n uw^n t \notin L$.

Soit $k = |vw|$; on en déduit $|sut| = (p - k)$. Par conséquent,

$$|sv^n uw^n t| = nk + p - k.$$

Il suffit de choisir n tel que $nk + p - k$ soit le produit de deux nombres supérieurs à 1 (et ne soit donc pas premier). Après quelques essais, on trouve que $n = p + 1$ est un bon choix. En effet,

$$(p+1)k + p - k = pk + p = p(k+1)$$

est le produit de deux nombres supérieurs à 1, car

- $p > 1$, par définition des nombres premiers.
- $k > 0$, puisque $v \neq \lambda$ ou $w \neq \lambda$.

Le langage $\{a^p : p \in \mathbf{N} \wedge p \text{ est premier}\}$ n'est donc pas un langage non contextuel.

Problème 1, page 161. Construisez une table d'analyse $LL(1)$ pour la grammaire

$$S \to aSc \qquad S \to b$$

Solution. La table demandée est la suivante.

	a	b	c	FIN
S	aSc	b	*erreur*	*erreur*

Problème 2, page 162. Soit la grammaire non contextuelle

$$G = (\{E, F, T\}, \{a, b, +, -, (,)\}, E, R)$$

où R est l'ensemble des règles suivantes.

$$\begin{array}{lll} E \to TF & F \to +TF & T \to (E) \\ & F \to -TF & T \to a \\ & F \to \lambda & T \to b \end{array}$$

Cette grammaire génère des expressions arithmétiques (d'où le E comme symbole de départ). Le F génère les « fins d'expressions » et le T génère les termes.

(a) Construisez une table d'analyse $LL(1)$ pour la grammaire G.

(b) Soient les séquences $u = ((a + b) - (a - b))$ et $v = (a + b)(a - b)$. Utilisez u et v comme données d'entrée pour la procédure LL(1) de la page 158. Donnez les informations demandées dans le programme. La table $LL(1)$ utilisée par le programme est évidemment celle que vous avez trouvée en (a).

(c) Dites si u et v sont acceptées ou non par le programme (c'est-à-dire si $u, v \in L(G)$). Pour chaque séquence acceptée, donnez la dérivation impliquée par l'exécution du programme.

Solution.

(a) La table d'analyse $LL(1)$ pour la grammaire G est :

	a	b	$+$	$-$	$($	$)$	FIN
E	TF	TF	TF	TF	TF	TF	TF
F	λ	λ	$+TF$	$-TF$	λ	λ	λ
T	a	b	*erreur*	*erreur*	(E)	*erreur*	*erreur*

Ce n'est pas la seule table qui soit correcte : il y a plusieurs entrées qui peuvent être remplacées par *erreur*. Par exemple, c'est le cas de la cellule $(E, +)$; en effet, comme $ET \to F$ est la seule dérivation possible à partir de E, et comme aucune dérivation à partir de T n'introduit un $+$, on voit qu'un E sur la pile et un $+$ à l'entrée conduisent nécessairement à une erreur. Mais il est inutile d'anticiper et il suffit de donner la réponse ci-dessus : puisque $E \to TF$ est la seule production applicable à E, on met un TF dans chaque case de la ligne E. Le contenu des cellules $(F, +)$ et $(F, -)$ est déterminé par les productions $F \to +TF$ et $F \to -TF$. Dans les autres cases de la ligne F, il suffit de mettre λ, puisque la seule production qui peut mener à une acceptation est $F \to \lambda$: par exemple, dans le cas de l'entrée (F, a), cela revient à dépiler F pour voir s'il y aurait un a en-dessous. Etc.

(b) Pour la séquence u, l'exécution du programme donne :

	Pile	Symbole	À lire
1	E	$($	$(a+b)-(a-b))$FIN
2	TF	$($	$(a+b)-(a-b))$FIN
3	$(E)F$	$($	$(a+b)-(a-b))$FIN
4	$E)F$	$($	$a+b)-(a-b))$FIN
5	$TF)F$	$($	$a+b)-(a-b))$FIN
6	$(E)F)F$	$($	$a+b)-(a-b))$FIN
7	$E)F)F$	a	$+b)-(a-b))$FIN
8	$TF)F)F$	a	$+b)-(a-b))$FIN
9	$aF)F)F$	a	$+b)-(a-b))$FIN
10	$F)F)F$	$+$	$b)-(a-b))$FIN
11	$+TF)F)F$	$+$	$b)-(a-b))$FIN
12	$TF)F)F$	b	$)-(a-b))$FIN
13	$bF)F)F$	b	$)-(a-b))$FIN
14	$F)F)F$	$)$	$-(a-b))$FIN
15	$)F)F$	$)$	$-(a-b))$FIN
16	$F)F$	$-$	$(a-b))$FIN
17	$-TF)F$	$-$	$(a-b))$FIN
18	$TF)F$	$($	$a-b))$FIN
19	$(E)F)F$	$($	$a-b))$FIN
20	$E)F)F$	a	$-b))$FIN
21	$TF)F)F$	a	$-b))$FIN
22	$aF)F)F$	a	$-b))$FIN
23	$F)F)F$	$-$	$b))$FIN
24	$-TF)F)F$	$-$	$b))$FIN
25	$TF)F)F$	b	$))$FIN
26	$bF)F)F$	b	$))$FIN
27	$F)F)F$	$)$	$)$FIN
28	$)F)F$	$)$	$)$FIN
29	$F)F$	$)$	FIN
30	$)F$	$)$	FIN
31	F	FIN	
32		FIN	

Pour la séquence v, l'exécution du programme donne :

	Pile	Symbole	À lire
1	E	(	$a+b)(a-b)$FIN
2	TF	(	$a+b)(a-b)$FIN
3	$(E)F$	(	$a+b)(a-b)$FIN
4	$E)F$	a	$+b)(a-b)$FIN
5	$TF)F$	a	$+b)(a-b)$FIN
6	$aF)F$	a	$+b)(a-b)$FIN
7	$F)F$	$+$	$b)(a-b)$FIN
8	$+TF)F$	$+$	$b)(a-b)$FIN
9	$TF)F$	b	$)(a-b)$FIN
10	$bF)F$	b	$)(a-b)$FIN
11	$F)F$	)	$(a-b)$FIN
12	$)F$	)	$(a-b)$FIN
13	F	(	$a-b)$FIN

(b) La séquence u est acceptée, mais pas v. La dérivation de u est la dérivation à gauche suivante :

$$
\begin{aligned}
&E \Rightarrow TF \Rightarrow (E)F \Rightarrow (TF)F \Rightarrow ((E)F)F \Rightarrow ((TF)F)F \\
&\Rightarrow ((aF)F)F \Rightarrow ((a+TF)F)F \Rightarrow ((a+bF)F)F \Rightarrow ((a+b)F)F \\
&\Rightarrow ((a+b)-TF)F \Rightarrow ((a+b)-(E)F)F \Rightarrow ((a+b)-(TF)F)F \\
&\Rightarrow ((a+b)-(aF)F)F \Rightarrow ((a+b)-(a-TF)F)F \\
&\Rightarrow ((a+b)-(a-bF)F)F \Rightarrow ((a+b)-(a-b)F)F \\
&\Rightarrow ((a+b)-(a-b))F \Rightarrow ((a+b)-(a-b))
\end{aligned}
$$

Problème 1, page 171. Soit la grammaire non contextuelle

$$G = (\{E, T\}, \{a, b, +, -, (,)\}, E, R)$$

où R est l'ensemble des règles suivantes.

$$\begin{array}{ll} E \to E + T & T \to (E) \\ E \to E - T & T \to a \\ E \to T & T \to b \end{array}$$

Cette grammaire génère des expressions arithmétiques (d'où le E comme symbole de départ). (Voyez aussi le problème 2 de la section 2.4, page 162). Ce qui suit est la table d'analyse $LR(1)$ correspondant à G.

	a	b	$+$	$-$	(	)	FIN	E	T
1	t 6	t 7			t 10			2	5
2			t 3	t 4			accepte		
3	t 6	t 7			t 10				8
4	t 6	t 7			t 10				9
5			$E \to T$	$E \to T$		$E \to T$	$E \to T$		
6			$T \to a$	$T \to a$		$T \to a$	$T \to a$		
7			$T \to b$	$T \to b$		$T \to b$	$T \to b$		
8			$E \to E + T$	$E \to E + T$		$E \to E + T$	$E \to E + T$		
9			$E \to E - T$	$E \to E - T$		$E \to E - T$	$E \to E - T$		
10	t 6	t 7			t 10			11	5
11			t 3	t 4		t 12			
12			$T \to (E)$	$T \to (E)$		$T \to (E)$	$T \to (E)$		

(a) Soient les séquences $u = ((a + b) - (a - b))$ et $v = (a + b)(a - b)$. Utilisez u et v comme données d'entrée pour la procédure LR(1) de la page 167. La table d'analyse $LR(1)$ utilisée par la procédure est la table ci-dessus. Donnez les informations demandées par le programme.

(b) Dites si u et v sont acceptées ou non par la procédure (c'est-à-dire si $u, v \in L(G)$). Pour chaque séquence acceptée, donnez la dérivation impliquée par l'exécution du programme.

Solution.

(a) Pour la séquence u, l'exécution du programme donne ce qui suit (la colonne « S » est la valeur de la variable Symbole et la colonne « J » la valeur de la variable Jeton).

	Pile	À lire	S	J	EntréeTable
1	1	$(a+b)-(a-b))$FIN	(	1	t 10
2	1 (10	$a+b)-(a-b))$FIN	(	10	t 10
3	1 (10 (10	$+b)-(a-b))$FIN	a	10	t 6
4	1 (10 (10 a 6	$b)-(a-b))$FIN	$+$	6	$T \to a$
5	1 (10 (10 T 5	$b)-(a-b))$FIN	$+$	5	$E \to T$
6	1 (10 (10 E 11	$b)-(a-b))$FIN	$+$	11	t 3
7	1 (10 (10 E 11 $+$ 3	$)-(a-b))$FIN	b	3	t 7
8	1 (10 (10 E 11 $+$ 3 b 7	$-(a-b))$FIN	)	7	$T \to b$
9	1 (10 (10 E 11 $+$ 3 T 8	$-(a-b))$FIN	)	8	$E \to E+T$
10	1 (10 (10 E 11	$-(a-b))$FIN	)	11	t 12
11	1 (10 (10 E 11) 12	$(a-b))$FIN	$-$	12	$T \to (E)$
12	1 (10 T 5	$(a-b))$FIN	$-$	5	$E \to T$
13	1 (10 E 11	$(a-b))$FIN	$-$	11	t 4
14	1 (10 E 11 $-$ 4	$a-b))$FIN	(	4	t 10
15	1 (10 E 11 $-$ 4 (10	$-b))$FIN	a	10	t 6
16	1 (10 E 11 $-$ 4 (10 a 6	$b))$FIN	$-$	6	$T \to a$
17	1 (10 E 11 $-$ 4 (10 T 5	$b))$FIN	$-$	5	$E \to T$
18	1 (10 E 11 $-$ 4 (10 E 11	$b))$FIN	$-$	11	t 4
19	1 (10 E 11 $-$ 4 (10 E 11 $-$ 4	$))$FIN	b	4	t 7
20	1 (10 E 11 $-$ 4 (10 E 11 $-$ 4 b 7	$)$FIN	)	7	$T \to b$
21	1 (10 E 11 $-$ 4 (10 E 11 $-$ 4 T 9	$)$FIN	)	9	$E \to E-T$
22	1 (10 E 11 $-$ 4 (10 E 11	$)$FIN	)	11	t 12
23	1 (10 E 11 $-$ 4 (10 E 11) 12	FIN	)	12	$T \to (E)$
24	1 (10 E 11 $-$ 4 T 9	FIN	)	9	$E \to E-T$
25	1 (10 E 11	FIN	)	11	t 12
26	1 (10 E 11) 12		FIN	12	$T \to (E)$
27	1 T 5		FIN	5	$E \to T$
28	1 E 2		FIN	2	*accepte*

Pour la séquence v, l'exécution du programme donne :

	Pile	À lire	S	J	EntréeTable
1	1	$a+b)(a-b)$FIN	(	1	t 10
2	1 (10	$+b)(a-b)$FIN	a	10	t 6
3	1 (10 a 6	$b)(a-b)$FIN	$+$	6	$T \to a$
4	1 (10 T 5	$b)(a-b)$FIN	$+$	5	$E \to T$
5	1 (10 E 11	$b)(a-b)$FIN	$+$	11	t 3
6	1 (10 E 11 $+$ 3	$)(a-b)$FIN	b	3	t 7
7	1 (10 E 11 $+$ 3 b 7	$(a-b)$FIN	)	7	$T \to b$
8	1 (10 E 11 $+$ 3 T 8	$(a-b)$FIN	)	8	$E \to E+T$
9	1 (10 E 11	$(a-b)$FIN	)	11	t 12
10	1 (10 E 11) 12	$a-b)$FIN	(	12	

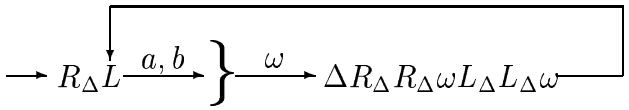
(b) La séquence u est acceptée, mais pas v. La dérivation de u est la dérivation à droite suivante :

$$\begin{aligned} &E \Rightarrow T \Rightarrow (E) \Rightarrow (E-T) \Rightarrow (E-(E)) \Rightarrow (E-(E-T)) \\ &\Rightarrow (E-(E-b)) \Rightarrow (E-(T-b)) \Rightarrow (E-(a-b)) \Rightarrow (T-(a-b)) \\ &\Rightarrow ((E)-(a-b)) \Rightarrow ((E+T)-(a-b)) \Rightarrow ((E+b)-(a-b)) \\ &\Rightarrow ((T+b)-(a-b)) \Rightarrow ((a+b)-(a-b)) \end{aligned}$$

5.4 Chapitre 3

Problème 1, page 187. En utilisant les machines de Turing élémentaires présentées à la section 3.2, construisez une machine de Turing qui passe de la configuration $\underline{\Delta}w\Delta\Delta\Delta\cdots$ à la configuration $\underline{\Delta}w\Delta v\Delta\Delta\Delta\cdots$, où $v, w \in \{a,b\}^*$ et v est la séquence w écrite à l'envers.

Solution. La machine de Turing demandée doit faire une copie inversée de la séquence d'entrée w. Elle ressemble beaucoup à celle de la figure 3.8 de [Brookshear89], qui fait une copie directe de w. Nous supposons ici que le ruban initial contient seulement w, puisque l'énoncé indique que la configuration initiale est $\underline{\Delta}w\Delta\Delta\Delta\cdots$. La machine suivante fait l'affaire. Remarquez qu'elle donne aussi le bon résultat si $w = \lambda$.



Problème 2, page 187. Le résultat de l'exécution de $\rightarrow RL$ peut-il être différent de celui de l'exécution de $\rightarrow LR$? Expliquez pourquoi.

Solution. Oui. Lorsque la tête de lecture est sur la première case du ruban, $\rightarrow RL$ laisse la tête au même endroit, alors qu'avec $\rightarrow LR$, la machine de Turing termine anormalement. Dans tous les autres cas, $\rightarrow RL$ et $\rightarrow LR$ ont le même effet (la tête de lecture ne bouge pas).

Problème 1, page 201. Montrez que si le langage L est acceptable par une machine de Turing, alors il existe une autre machine de Turing qui termine anormalement si et seulement si la séquence d'entrée qui lui est soumise appartient à L. (Suggestion : jetez un coup d'oeil au théorème 3.20.)

Solution. Soit M la machine de Turing qui accepte L ; on a $L = L(M)$. Nous supposons que M accepte en terminant normalement et rejette en bouclant ou en terminant anormalement. Supposons aussi que la configuration initiale soit $\underline{\Delta}w\Delta\Delta\Delta\cdots$. Il faut construire une machine de Turing M' telle que

M' termine anormalement si et seulement si $w \in L$,

c'est-à-dire

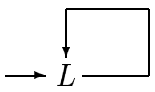
M' termine anormalement si et seulement si M accepte.

Soit Γ l'alphabet de ruban de M et supposons que $\# \notin \Gamma$. Choisissons $\Gamma' = \Gamma \cup \{\#\}$ comme alphabet de ruban de M'.

1. M' doit d'abord passer de la configuration $\underline{\Delta}w\Delta\Delta\cdots$ à $\#\underline{\Delta}w\Delta\Delta\cdots$ afin de marquer la première case du ruban avec le symbole $\#$; ceci lui permettra ensuite, lors de la simulation de M, de détecter les cas où M termine anormalement. La machine M' commence donc par

$$\longrightarrow R_\Delta S_R L_\Delta L \# R$$

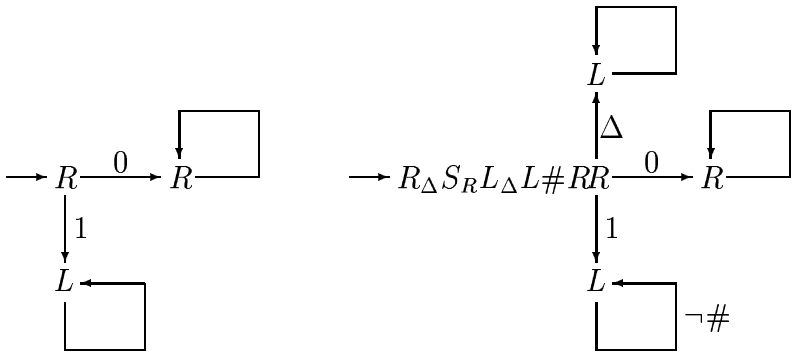
2. M' doit ensuite simuler M, en surveillant l'occurrence de deux situations particulières.
   (a) Si M' atteint l'état correspondant à l'état final de M, elle doit exécuter



   afin de terminer anormalement (puisque M accepte).
   (b) Si $\#$ devient le symbole courant, c'est que M termine anormalement. M' peut tout simplement s'arrêter (M' pourrait aussi boucler indéfiniment). Notez qu'il suffit de vérifier la présence du $\#$ après chaque mouvement de la tête vers la gauche.

Remarquez que si M exécute une boucle infinie, alors il en sera de même de M'.

Voici un exemple. Soit $\Gamma = \{0, 1, \Delta\}$. On a, à gauche, la machine M, et à droite, la machine M' correspondante.



Après le déplacement initial de la tête vers la droite,

(a) si symbole courant $= \Delta$, M s'arrête et M' termine anormalement ;

(b) si symbole courant $= 0$, M et M' bouclent toutes les deux ;

(c) si symbole courant $= 1$, M termine anormalement alors que M' déplace la tête de lecture vers la gauche et s'arrête lorsque la tête est sur le $\#$.

Problème 1, page 204. Construisez une grammaire à structures de phrase G telle que

$$L(G) = \{a^m b^n a^m b^n : m, n \in \mathbf{N}\}.$$

Solution. Construisons la grammaire $G = (\{S, A, B, D, F\}, \{a, b\}, S, R)$ progressivement. Avec les 3 règles

$$S \to AF \qquad A \to aAa \qquad A \to B$$

G génère les séquences de la forme $a^m B a^m F$, où $m \in \mathbf{N}$. En ajoutant les règles

$$B \to bBN \qquad B \to D$$

les séquences générées ont la forme $a^m b^n D N^n a^m F$, où $m, n \in \mathbf{N}$. Avec les règles

$$Na \to aN \qquad Da \to aD$$

les a passent à gauche de la sous-chaîne DN^n et les séquences générées ont la forme $a^m b^n a^m DN^n F$. Finalement, les règles

$$NF \to Fb \qquad DF \to \lambda$$

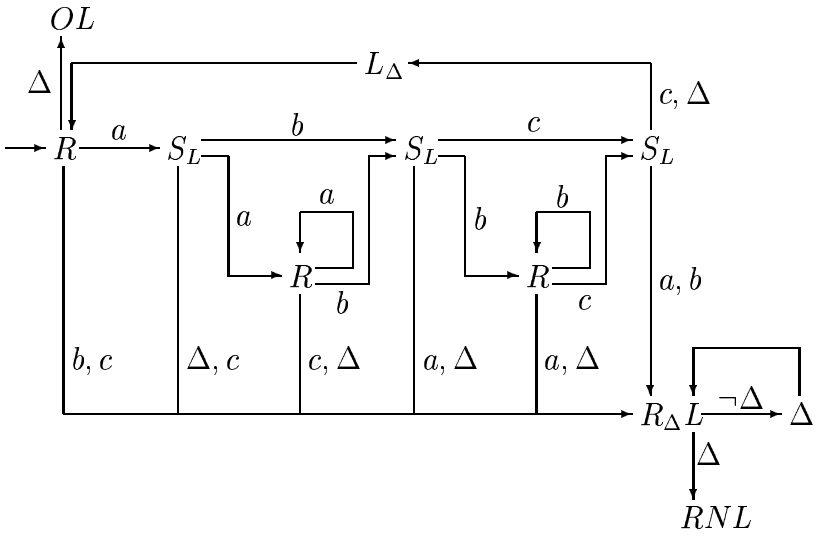
permettent d'obtenir les séquences de la forme désirée (et seulement celles-ci). La grammaire complète est donc

$$\begin{array}{lllll} S \to AF & A \to aAa & A \to B & B \to bBN & B \to D \\ Na \to aN & Da \to aD & NF \to Fb & DF \to \lambda & \end{array}$$

Problème 1, page 210. Soit $L = \{a^n b^n c^n : n \in \mathbf{N}\}$. Construisez une machine de Turing M qui décide L.

Solution. Il suffit de transformer quelque peu la machine de l'exemple 3.16. Faisons d'abord 2 remarques.

1. Si la machine s'arrête, c'est dans la configuration $\Delta\underline{\Delta}\Delta\Delta\cdots$, après avoir exécuté le R au coin supérieur gauche (voir l'exemple sus-mentionné). Il suffit alors de passer le contrôle à la sous-machine OL, pour accepter.
2. Tous les chemins qui mènent au R inférieur gauche (du même exemple) doivent conduire plutôt à une sous-machine (coin inférieur droit du diagramme ci-dessous) qui produit la configuration finale $\underline{\Delta}N\Delta\Delta\cdots$. Cette sous-machine utilise le fait qu'en aucun temps la séquence d'entrée n'est coupée en sous-séquences séparées par des Δ ; c'est pour cette raison que le R_Δ inférieur droit positionne la tête de lecture à droite de tout symbole différent de Δ qui pourrait rester sur le ruban. Il ne reste plus qu'à effacer ces symboles non blancs et à écrire le N.


OL
Δ
L_Δ
c, Δ
R
a
S_L
b
S_L
c
S_L
a
a
b
b
R
b
R
c
a, b
b, c
Δ, c
c, Δ
a, Δ
a, Δ
R_Δ
L
¬Δ
Δ
Δ
RNL

Chapitre 6

Autres commentaires

Preuve alternative de l'exemple 0.52

Montrons par induction ($\forall n \in \mathbf{N} : 2^{n+1} - 1 = \sum_{i=0}^{n} 2^i$). Soit $P(n)$ le prédicat

$$2^{n+1} - 1 = \sum_{i=0}^{n} 2^i.$$

Il faut montrer qu'il est vrai pour tout $n \in \mathbf{N}$.

- Base d'induction. Il faut montrer $P(n)$ pour la plus petite valeur de n, soit $n = 0$:

  $P(0)$
  $\Leftrightarrow$ $\langle$ Définition de P $\rangle$
  $2^{0+1} - 1 = \sum_{i=0}^{0} 2^i$
  $\Leftrightarrow$ $\langle$ Arithmétique et définition de $\sum$ $\rangle$
  $1 = 2^0$
  $\Leftrightarrow$ $\langle$ Arithmétique $\rangle$
  vrai

- Étape d'induction. Supposons que $P(k)$ est vrai pour un $k \geq 0$ arbitraire et montrons $P(k+1)$. (Le prédicat $P(k)$ est appelé *l'hypothèse d'induction.*)

  $\sum_{i=0}^{k+1} 2^i$
  $=$ $\langle$ Extraction d'un terme de $\sum$ $\rangle$
  $2^{k+1} + \sum_{i=0}^{k} 2^i$
  $=$ $\langle$ Hypothèse d'induction $\rangle$
  $2^{k+1} + 2^{k+1} - 1$
  $=$ $\langle$ Arithmétique : $a + a = 2a$ $\rangle$

$$
\begin{array}{ll}
& 2 \times 2^{k+1} - 1 \\
= & \qquad \langle \text{ Arithmétique : } a^m \times a^n = a^{m+n} \rangle \\
& 2^{k+2} - 1 \\
= & \qquad \langle \text{ Arithmétique } \rangle \\
& 2^{(k+1)+1} - 1
\end{array}
$$

L'ensemble des rationnels est dénombrable

Un nombre rationnel non négatif (un élément de $\mathbf{Q}^+$) est un nombre qui peut s'exprimer comme le rapport de deux nombres naturels p et q sans facteur commun autre que 1. Il ne peut donc y avoir plus de nombres rationnels non négatifs que de paires de nombres naturels (p, q), c'est-à-dire qu'on a $|\mathbf{Q}^+| \leq |\mathbf{N} \times \mathbf{N}|$. Or, nous savons que $\mathbf{N} \times \mathbf{N}$ est dénombrable (exemple 0.70 des notes). Par conséquent $\mathbf{Q}^+$ est dénombrable. Il est ensuite facile de montrer que $\mathbf{Q}$ est dénombrable (essayez).

Sur le mélange des styles de preuve

Plusieurs tentent de mélanger le style de preuve utilisé dans les notes de cours (on parle de logique équationnelle) et le style utilisé dans le cours de *Structures discrètes* (appelé *style de Hilbert*). C'est ainsi que j'ai vu des « preuves » qui ressemblent à la suivante :

$$
\begin{array}{ll}
1. & x > 4 \ \wedge \ x < 9 \\
\Rightarrow & \qquad \langle \text{ Par 1 et la loi } p \wedge q \Rightarrow p \text{ (ou autre justification similaire) } \rangle \\
2. & x > 4 \\
\Rightarrow & \qquad \langle \text{ Par 1 et la loi } p \wedge q \Rightarrow q \text{ (ou autre justification similaire) } \rangle \\
3. & x < 9
\end{array}
$$

Les lignes 2 et 3 sont liées par une implication. Elles affirment que $x > 4 \Rightarrow x < 9$, ce qui bien sûr est incorrect (en prenant $x = 10$, par exemple, on voit que l'implication est fausse). L'erreur vient de la tentative d'imiter le style de Hilbert en utilisant le format des notes de cours. Avec le style de Hilbert, ce qui suit est correct :

$$
\begin{array}{lll}
1. & x > 4 \ \wedge \ x < 9 & \\
2. & x > 4 & \text{Par 1 et la loi } p \wedge q \Rightarrow p \text{ (ou justification similaire)} \\
3. & x < 9 & \text{Par 1 et la loi } p \wedge q \Rightarrow q \text{ (ou justification similaire)}
\end{array}
$$

On peut ensuite utiliser les lignes 2 et 3 dans le reste de la preuve.

Le style de la logique équationnelle est ce qui se rapproche le plus de l'algèbre du secondaire. Supposons $m, n \in \mathbf{N}$. Vous comprenez bien la suite d'équations

$$
\begin{array}{ll}
 & (m+1)^2 + 2m \\
= & \qquad \langle\ (m+1)^2 = m^2 + 2m + 1\ \rangle \\
 & m^2 + 2m + 1 + 2m \\
= & \qquad \langle\ \text{Associativité et commutativité de } +\ \ \&\ \ 2m + 2m = 4m\ \rangle \\
 & m^2 + 4m + 1
\end{array}
$$

et il ne vous viendrait pas à l'idée (j'espère) d'écrire

$$
\begin{array}{ll}
1. & m + n \\
\leq & \qquad \langle\ \text{Par 1 et la loi } m + n \leq m\ \rangle \\
2. & m \\
\leq & \qquad \langle\ \text{Par 1 et la loi } m + n \leq n\ \rangle \\
3. & n
\end{array}
$$

car il est clair qu'on ne peut affirmer $m \leq n$. Pourquoi alors faire de telles transformations pour des expressions booléennes ? Elles devraient être traitées d'une manière semblable aux expressions arithmétiques.

Petite note historique en passant : la logique équationnelle utilisée dans le cours est semblable à celle du livre de Gries et Schneider (voyez la bibliographie, à la page 307 des notes de cours). Ce sont des informaticiens qui sont à l'origine de cette logique. Il y a en particulier le fameux Edsger W. Dijkstra, qui a lancé le premier « à bas le goto », ainsi que Backhouse, Gries et plusieurs Hollandais. Ces chercheurs travaillent dans le domaine de la dérivation de programmes à partir de spécifications mathématiques et ils ont senti le besoin d'une logique qui permettrait de présenter plus clairement les dérivations.

Si vous regardez les livres de mathématiques, vous constaterez que la grande majorité d'entre eux présentent leurs preuves en langue naturelle et qu'ils n'utilisent aucun des styles formels comme le style de Hilbert ou le style dit de *déduction naturelle*. Par contre, depuis que la logique équationnelle a été introduite, de plus en plus d'articles et de livres utilisent ce style. On peut espérer qu'un jour les livres de mathématiques utilisés dans l'enseignement adopteront ce style, de sorte que la manipulation d'expressions booléennes devienne aussi familière que la manipulation de polynômes, par exemple.

Bibliographie

[Brookshear89] J. Glenn Brookshear. *Formal languages, automata, and complexity.* The Benjamin/Cummings Publishing Company, Inc., 1989.

[Gries93] David Gries et Fred B. Schneider. *A logical approach to discrete Math.* Springer-Verlag, New York, 1993.

[Salomaa89] Arto Salomaa. *Introduction à l'informatique théorique : calculabilité & complexité.* A. Colin, Paris, 1989.

Index