République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A. Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

Support de cours

Théorie des Langages

Cours et exercices destinés aux étudiants de deuxième année informatique.

Réalisé par

Dr. Sofiane AISSANI

Année 2019

Table des matières

Table des figures

Liste des abréviations

ABL	Automate à Bornes Linéaires
AEFG	Automate à États Finis Généralisé
AEFM	Automate à États Finis Minimal
AEFPG	Automate à États Finis Partiellement Généralisé
AEFSD	Automate à États Finis Simple et Déterministe
AEFSND	Automate à États Finis Simple et Non Déterministe
AEFS	Automate à États Finis Simple
AEF	Automate à États Finis
AàPD	Automate à Pile Déterministe
AàPND	Automate à Pile Non Déterministe
AàP	Automate à Pile
FNC	Forme Normale de Chomsky
FNG	Forme Normale de Greibach
MTD	Machine de Turing Déterministe
MTND	Machine de Turing Non Déterministe
MT	Machine de Turing
ThL	Théorie des Langages

Préface

Ce document est un support de cours du module "théorie des langages", destiné aux étudiants de deuxième année informatique. C'est le fruit d'une expérience de quatre ans d'enseignement. Toutefois, malgré l'effort que nous avons fourni pour sa réalisation, il reste largement perfectible et probablement non exempt d'erreurs, toute remarque éventuelle qui pourrait l'améliorer sera la bienvenue.

Pour construire ce cours, nous nous sommes principalement inspiré des supports suivants :

- MAZOUZ S., "Theorie des langages", notes de cours, Université des Sciences et de la Technologie Houari Boumediene, Alger, 2005 ;
- BERLIOUX P., LEVY M., "Théorie des langages", Notes de cours, École nationale supérieure d'informatique et de mathématiques appliquées, Grenoble, 2018 ;
- CHOMSKY N., "Three models for the description of language". IRE-Transactions on Information Theory : 113124, 1956 ;
- BAR-HILLEL Y. MICHA A., SHAMIR E., "On formal properties of simple phrase structure grammars", Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikations for schung :143-172, 1961 ;
- PERIN M., "Modèles de calcul-Machines de Turing", notes de cours, Polytech-Université Grenoble-Alpes, Grenoble, 2018.
- TURING A., "On Computable Numbers, with an Application to the Entscheidungsproblem", proceedings of the London Mathematical Society :230-265, 1936.
- http://www.futura-sciences.com/magazines/mathematiques/infos/ dossiers/d/ mathematiques-infini-il-paradoxal-mathematiques-1590/page/9, consulté en 2019.

La théorie des langages formels est une branche commune entre les mathématiques, la linguistique et l'informatique fondamentale. En mathématique, elle se base sur la théorie des ensembles, qualifiée de "paradis des mathématiciens" par David Hilbert, d'ailleurs c'est le pré-requis le plus important pour entamer ce cours. En linguistiques, elle est utilisée comme moyen pour comprendre les régularités syntaxiques des langages

naturels. Mais l'aspect qui nous intéresse le plus dans ce cours, c'est la branche informatique. En effet, le contenu de ce cours vise principalement deux objectifs informatiques : d'une part, un objectif académique qui est lui même divisé en deux sous-objectifs, le premier consiste en l'acquisition des notions fondamentales qui serviront de base aux cours du module de compilation. Le second concerne l'assimilation du fonctionnement de quelques types de machines abstraites, notamment les machines de Turing, qui non seulement permettent la définition formelle de la notion de calculabilité, mais surtout servent de base à la théorie de la complexité algorithmique. D'autre part, un objectif scientifique ; en effet, ce cours apporte aux étudiants une panoplie d'outils de modélisation comme les automates à états finis ou encore les expressions régulières qui sont très utilisés dans des problèmes de recherches scientifiques, notamment en sécurité informatique.

Ce support de cours est subdivisé en quatre chapitres, nous définissons d'abord, dans chacun, les notions relatives au chapitre que nous illustrons par des exemples, ensuite nous posons une série d'exercices consistante qui permet d'approfondir les différentes notions du cours.

Dans le premier chapitre, nous définissons les notions fondamentales de la théorie des langages. Ainsi, nous commençons par des définitions élémentaires comme les langages et les opérations entre les langages. Ensuite, nous étudions la notion de grammaire formelle, puis nous terminons par la classification de Noham Chomsky des grammaires et des langages.

Le second chapitre est consacré aux langages réguliers (rationnels), et aux machines abstraites les reconnaissant, en l'occurrence les automates à états finis.

Le troisième chapitre concerne les langages dits à contexte libre, et les machines abstraites les reconnaissant, c'est-à dire les automates à pile.

Le quatrième et dernier chapitre joint les langages décidables et semi-décidables ainsi que que leurs machines abstraites correspondantes (automate à bornes linéaires et machines de Turing), nous avons regroupé ces deux types de langage dans le même chapitre, car la différence entre les deux machines abstraites les reconnaissant consiste seulement dans la taille du ruban dans lequel le mot à reconnaître est inscrit.

Chapitre 1

Notions fondamentales de la théorie des langages

Sommaire

Le présent chapitre est principalement constitué de quatre parties. Dans la première, nous définissons les notions de bases relatives aux langages formels. Dans la seconde nous nous intéressons aux opérations les plus connues qu'on peut effectuer sur des langages. La troisième partie est dédiée au grammaires syntagmatiques et à la classification de Chomsky qui se base sur la forme des règles de production. Puis, nous terminons par une série d'exercices pour approfondir la maîtrise des différents concepts étudiés.

1.1 Définitions et notions de base

1.1.1 Définition d'un alphabet

Un alphabet X est un ensemble fini et non vide, les éléments de cet ensemble sont appelés des lettres ou des symboles.

1.1.2 Définition d'un mot

On appelle mot sur un alphabet X toute suite finie, éventuellement vide, d'éléments de X.

Notations :

- Le mot vide est noté "ε".
- L'ensemble des mots sur un alphabet X est noté X^*.
- X^+ est l'ensemble des mots non vides.

1.1.3 Définition de la concaténation

Soient ω_1 et ω_2 deux mots de X^*, on définit la concaténation comme la juxtaposition de ω_1 et ω_2 noté : $\omega_1.\omega_2$.
Formellement, $\omega_1.\omega_2$ est défini comme suit :

- $\varepsilon.\omega_1 = \omega_1.\varepsilon = \omega_1$
- $\omega_1.\omega_2 = (a_1...a_n).(b_1...b_m) = a_1...a_nb_1...b_m$ où les a_i et les b_j représentent les lettres constituant respectivement ω_1 et ω_2.

Théorème : $(X^*, ., \varepsilon)$ est un monoïde. Pour démontrer ce théorème, il suffit de montrer que :

- La concaténation est une loi de composition interne dans X^* ;

- La concaténation est une loi de composition associative ;
- Le mot vide est l'élément neutre de la concaténation.

1.1.4 Définition de la longueur d'un mot

On appelle longueur d'un mot ω, la somme des occurrences des différentes lettres le constituant, elle est souvent notée $|\omega|$.
Formellement, si $a \in X$ et $\omega \in X^*$, on définit la longueur par induction comme suit :

- $|\varepsilon| = 0$;
- $|a.\omega| = 1 + |\omega|$.

Notation : On utilise la notation $|\omega|_a$ qui représente le nombre de a dans le mot ω.

Théorème : La longueur est un homomorphisme de monoïdes de $(X^*, ., \varepsilon)$ dans (N,+,0).

Pour démontrer ce théorème, il suffit de montrer que :

- $(X^*, ., \varepsilon)$ est un monoïde ;
- (N,+,0) est un monoïde ;
- "Longueur" est une application ;
- $|\omega_1.\omega_2|=|\omega_1| + |\omega_2|$ (par récurrence).

1.1.5 Définition du miroir d'un mot

Soient $a_1, a_2..., a_n$ n lettres d'un alphabet X. On appelle miroir du mot $\omega = a_1a_2...a_n$ le mot noté $\omega^R = a_n...a_2a_1$.

Formellement, le miroir est défini par induction comme suit :

- $\varepsilon^R = \varepsilon$;
- $a^R = a$, avec $a \in X$;
- $(a.\omega)^R = \omega^R.a$, avec $a \in X$.

1.1.6 Définition de la puissance d'un mot

Soit ω un mot de X^*. On définit la puissance de ω comme étant la juxtaposition du mot ω plusieurs fois.

Formellement, on définit la puissance par induction comme suit :

- $\omega^0 = \varepsilon$;

- $\omega^1 = \omega$;
- $\omega^{n+1} = \omega^n.\omega$.

1.1.7 Factorisation

Soit X un alphabet, soient v et w deux mots de X^*, on dit que :

- v est un sous-mot (facteur) de w si et seulement s'il existe deux mots u_1 et u_2 appartenant à X^* tel que : $w = u_1 v u_2$;
- v est un facteur propre de w si et seulement si $u_1 \neq \varepsilon$ et $u_2 \neq \varepsilon$;
- v est un facteur gauche de w si et seulement si $u_1 = \varepsilon$;
- v est un facteur droit de w si et seulement si $u_2 = \varepsilon$.

Exemples :

- "bon" est un facteur gauche du mot "bonjour".
- "jour" est un facteur droit du mot "bonjour".
- "on" est un facteur propre du mot "bonjour".

1.1.8 Définition d'un langage formel

On appelle langage formel défini sur un alphabet X, tout sous-ensemble (fini ou infini) de X^*.

Remarques :

- Un langage fini est un langage contenant un nombre fini de mots ;
- Le langage vide ne contient aucun mot ;
- Un langage est dit propre s'il ne contient pas le mot vide ;
- Un langage est infini s'il n'est ni vide ni fini. Certains langages infinis (langages semi-décidables) peuvent être décrits par un ensemble de règles, appelé une grammaire formelle. Il existe d'autres langages infinis pour lesquels il n'existe aucun moyen de description, on les appelle des langages indécidables.

Exemples de langages :

- $L_1 = \{ab, a, ba, bb\}$;
- $L_2 = \{\omega \in \{a, b\}^* \,/\, |\omega| \geqslant 3\}$;
- $L3 = \{\omega \in \{a, b\}^* \,/\, |\omega| \equiv 0\,[5]\}$;
- $L_4 = \{\omega \in \{a, b\}^* \,/\, |\omega|_a \equiv 0\,[3]\}$;

- $L_5 = \{a^n / n \geqslant 0\}$;
- $L_6 = \{a^i b^j / i \geqslant 0, j \geqslant 1\}$;
- $L_7 = \{a^i b^i / i \geqslant 0\}$;
- $L_8 = \{a^i b^j a^j / i \geqslant j > 1\}$.

1.2 Opérations sur les langages

Les langages sont des ensembles, par conséquent on peut leur appliquer toutes les opérations appliquées sur les ensembles ; toutefois, il existe des opérations qui leurs sont spécifiques, il s'agit d'une extension des opérations définies sur les mots.

Soient L, L_1 et L_2 trois langages définis respectivement sur les deux alphabets X, X_1 et X_2 :

- L'union : $L_1 \cup L_2 = L_1 + L_2 = \{\omega / \omega \in L_1 \vee \omega \in L_2\}$;
- L'intersection : $L_1 \cap L_2 = \{\omega / \omega \in L_1 \wedge \omega \in L_2\}$;
- Le complément : $\overline{L_1} = \{\omega / \omega \in {X_1}^* \wedge \omega \notin L_1\}$;
- La concaténation des langages : $L_1.L_2 = \{\omega = \omega_1.\omega_2 / \omega_1 \in X_1 \wedge \omega_2 \in X_2\}$;
- La puissance concaténative : $L^n = L.L.....L$(n fois L). On peut le définir par induction comme suit : $L^0 = \{\varepsilon\}$, $L^1 = L$ et $L^n = L.L^{n-1}$;
- La fermeture itérative ou l'étoile de Kleen : $L^* = L^0 \cup L^1 \cup ... \cup L^n$(n tend vers l'infini) ;
- La fermeture itérative propre (l'étoile propre) : $L^+ = L^1 \cup L^2 \cup ... \cup L^n$ (n tend vers l'infini) ;
- Le miroir de L : $L^R = \left\{\omega / \exists u \in L / \omega = u^R\right\}$.

Exemples : Soient L_1, L_2 et L_3 trois langages définis par : $L_1 = \{\varepsilon, aa\}$, $L_2 = \{a^i b^j / i, j \geqslant 0\}$ et $L_3 = \{ab, b\}$.
Calculer : $L_1.L_2, L_1.L_3, L_1 \cup L_2, L_2 \cap L_3, L_1^{10}, L_1^*, L_1^+, L_2^R$.

Solutions :

- $L_1.L_2 = L_2$;
- $L_1.L_3 = \{ab, b, aaab, aab\}$;
- $L_1 \cup L_2 = L_2$;
- $L_2 \cap L_3 = L_3$;
- $L_1^{10} = \{a^{2n} / 10 \geqslant n \geqslant 0\}$;
- $L_1^* = L_1^+ = \{a^{2n} n \geqslant 0\}$;

- $L_2^R = \{b^i a^j / i, j \geqslant 0\}$.

1.2.1 Propriétés

En plus des propriétés des opérateurs ensemblistes qui sont toujours valables pour les langages, nous définissons les propriétés supplémentaires suivantes :

- La concaténation des langages n'est pas idempotante. cela signifie que $\exists L / L.L \neq L$;
- La concaténation des langages est associative ;
- La concaténation des langages n'est pas commutative ;
- La concaténation des langages est distributive par rapport à l'union ;
- La concaténation des langages n'est pas distributive par rapport à l'intersection ;
- $L^* = (L^*)^*$;
- $L^* = L^*.L^*$;
- $(L_1 + L_2)^* = (L_1^*.L_2^*)^* = (L_1^* + L_2^*)^*$.

1.3 Grammaire syntagmatique

Les grammaires syntagmatiques (grammaires formelles) sont un cas particulier d'objets mathématiques, elles sont aussi appelées systèmes de substitution ou de remplacement.

1.3.1 Définition formelle

Une grammaire est un quadruplé G=(T,N,S,P) où :

- T : est l'ensemble des symboles terminaux (les lettres constituant les mots du langage) ;
- N : est l'ensemble des symboles non terminaux ;
- S : est l'axiome (point de départ), S ∈N ;
- P : est l'ensemble des règles de production, où une règle de production est sous la forme $\alpha \rightarrow \beta$, avec $\alpha \in (T \cup N)^* N (T \cup N)^*$ et $\beta \in (T \cup N)^*$.

Notation : Lorsque plusieurs règles de production d'une grammaire ont une même forme en partie gauche, on pourra les factoriser en séparant les parties droites par des traits verticaux. Si par exemple A→aA et A→ε, alors on écrira directement A→aA|ε.

1.3.2 Définition d'une dérivation directe (immédiate)

Soit une grammaire G=(T,N,S,P), on dit qu'un mot $m_1 \in (T \cup N)^* N (T \cup N)^*$ dérive directement un mot $m_2 \in (T \cup N)^*$ si et seulement s'il existe une règle de production $\alpha \to \beta \in P$ et $\exists \delta, \gamma \in (T \cup N)^*$ tel que : $m_1 = \delta\alpha\gamma$ et $m_2 = \delta\beta\gamma$. On écrit alors $m_1 \Rightarrow m_2$ ou-bien $m_1 \vdash m_2$.

1.3.3 Définition d'une dérivation indirecte

Un mot m_2 peut être dérivé d'un mot m_1 si et seulement si $m2$ peut être obtenu à partir de m_1 en 0, 1 ou plusieurs dérivations directes. On note : $m_1 \Rightarrow^* m_2$ ou-bien $m_1 \vdash^* m_2$.

1.3.4 Définition du langage généré par une grammaire

Soit une grammaire G=(T,N,S,P), le langage L(G) généré (engendré) par la grammaire G est défini par : $L(G) = \{\omega / S \Rightarrow^* \omega, \omega \in T^*\}$.

Le langage généré par G contient exactement les mots dérivables à partir de l'axiome S par les règles de P et qui ne contiennent que des éléments de T.

1.3.5 Équivalence entre grammaires

Deux grammaires G_1 et G_2 sont équivalentes si et seulement si elles engendrent exactement le même langage.

Exemples : Trouver des grammaires qui génèrent les langages suivants.

- $L_1 = \{ab, a, ba, bb\}$;
- $L_2 = \{a^n / n \geqslant 0\}$;
- $L_3 = \{a^{2n} / n \geqslant 1\}$;
- $L_4 = \{a^i b^j / i \geqslant 0, j \geqslant 1\}$;
- $L_5 = \{a^i b^i / i \geqslant 0\}$;
- $L_6 = \{a^i b^j / i \geqslant j > 0\}$;
- $L_7 = \left\{\omega\omega^R / \omega \in \{a, b\}^*\right\}$.

Solutions :

- Solution G_1 : T=$\{a, b\}$, N= $\{S\}$, l'axiome est S et les règles de production sont les suivantes :
  S→a|ab|ba|bb
- Solution 1 G_2 : T=$\{a\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aS|ε
- Solution 2 G_2 : T=$\{a\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→Sa|ε
- Solution 1 G_3 : T=$\{a\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aaS|aa
- Solution 2 G_3 : T=$\{a\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aSa|aa
- Solution 3 G_3 : T=$\{a\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→Saa|aa
- Solution 4 G_3 : T=$\{a\}$, N= $\{S, A\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aA|aa
  A→aS
- Solution 1 G_4 : T=$\{a, b\}$, N= $\{S, A, B\}$, l'axiome : S et les règles de production sont les suivantes :
  S→AB
  A→aA|ε
  B→bB|b
- Solution 2 G_4 : T=$\{a, b\}$, N= $\{S, B\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aS|B
  B→bB|b
- Solution 3 G_4 : T=$\{a, b\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aS|Sb |b

- Solution 1 G_5 : T=$\{a, b\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aSb|ε
- Solution 2 G_5 : T=$\{a, b\}$, N= $\{S, B\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aB|ε
  B→Sb
- Solution 3 G_5 : T=$\{a, b\}$, N= $\{S, A\}$, l'axiome : S et les règles de production sont les suivantes :
  S→Ab|ε
  B→aS
- Solution 1 G_6 : T=$\{a, b\}$, N= $\{S, A, B\}$, l'axiome : S et les règles de production sont les suivantes :
  S→AB
  A→aA |ε
  B→aBb|ab
- Solution 2 G_6 : T=$\{a, b\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aS|aSb|ab
- Solution 3 G_6 : T=$\{a, b\}$, N= $\{S, A, B\}$, l'axiome : S et les règles de production sont les suivantes :
  S→Ab|B
  A→aS
  B→aB|ab
- Solution G_7 : T=$\{a, b\}$, N= $\{S\}$, l'axiome : S et les règles de production sont les suivantes :
  S→aSa|bSb |ε

1.3.6 Classification des grammaires et des langages

Noam Chomsky a proposé, en 1956, une classification des grammaires selon leur complexité en se basant sur la forme des règles. Chomsky a ainsi proposé quatre classes (hiérarchiques) de grammaires (et de langages), de sorte qu'une grammaire de type i génère un langage de type j, avec $j \geqslant i$.

1.3.6.1 Grammaire de type 3 (régulière)

Une grammaire est de type 3 si et seulement si elle est régulière à droite ou régulière à gauche.

Définition d'une grammaire régulière à droite : Soit la grammaire G=(T,N,S,P), G est dite régulière à droite si et seulement si toutes ses règles de production ont l'une des formes suivantes : $A \to \omega B$ ou $A \to \omega$ avec A,B∈N et $\omega \in T^*$.

Définition d'une grammaire régulière à gauche : Soit la grammaire G=(T,N,S,P), G est dite régulière à gauche si et seulement si toutes ses règles de production ont l'une des formes suivantes : $A \to B\omega$ ou $A \to \omega$ avec A,B∈N et $\omega \in T^*$.

Les langages de type 3 (réguliers) : Ce sont les langages qui peuvent être définis par des grammaires de type 3. Cet ensemble de langages peut être reconnu par les automates à états finis (AEF).

1.3.6.2 Grammaire de type 2 (hors contexte)

Soit la grammaire G=(T,N,S,P), G est dite de type 2 si et seulement si toutes ses règles sont sous la forme : $A \to B$ avec $A \in$ N et $B \in (T \cup N)^*$.

Les langages de type2 (hors contexte ou algébrique) : Ce sont les langages qui peuvent être définis par des grammaires de type 2. Cet ensemble de langages peut être reconnu par les automates à piles (AàP).

1.3.6.3 Grammaire de type 1 (contextuelle)

Définition d'une grammaire monotone : Une grammaire G=(T,N,S,P) est dite monotone si et seulement si toutes ses règles de production sont sous la forme $A \to B$ avec $A, B \in (T \cup N)^*$ et $|A| \leq |B|$. Cette définition interdit en particulier d'engendrer le mot vide, pour remédier à ce problème, dans une grammaire de type 1, on permet la règle où l'axiome génère le mot vide.

Définition d'une grammaire contextuelle : Soit la grammaire G=(T,N,S,P), G est dite contextuelle si et seulement si toutes ses règles sont sous la forme : $\alpha A\gamma \to \alpha\omega\gamma$

avec $A \in$ N et $\alpha, \gamma, \omega \in (T \cup N)^*$. Et seul l'axiome peut générer le mot vide. On remarque,par construction, que toute grammaire contextuelle est monotone.

Théorème : Pour tout langage L engendré par une grammaire monotone G, il existe une grammaire contextuelle qui engendre L.

Remarque :

- Les grammaires contextuelles et monotones engendrent exactement les mêmes langages, ce sont les langages de type 1.

Les langages de type1 (contextuels ou sensibles au contexte) : Ce sont les langages qui peuvent être définis par des grammaires de type 1. Cet ensemble de langages peut être reconnu par les automates à bornes linéaires(ABL).

1.3.6.4 Grammaire de type 0

On appelle grammaire de type 0 une grammaire syntagmatique dans laquelle la forme des productions est non-contrainte. $A \to B$ avec $A \in (T \cup N)^* N (T \cup N)^*$ et $B \in (T \cup N)^*$.

Les langages de type 0 (semi-décidables ou récursivement énumérables) : Ce sont les langages qui peuvent être définis par des grammaires de type 0. Cet ensemble de langages peut être reconnu par les machines de Turing (MT).

Comme cela apparaît sur la figure 1.1, il existe une relation d'inclusion stricte entre les quatre types de grammaires, mais en pratique quand on dit qu'une grammaire est de type i, cela signifie implicitement qu'elle n'est pas de type i+1 ; on s'intéresse au plus petit type en terme d'inclusion.

Remarque : Les langages hors contexte (générés par les grammaires de type 3 ou de type 2) jouent un rôle particulièrement important, car des analyseurs syntaxiques performants ont été développés pour cette catégorie de grammaires, cependant la plus part des langages de programmations ne peuvent être entièrement décrits par une grammaire à contexte libre. Les parties sensibles au contexte (règles de compatibilités des types, les paramètres lors de l'appel d'une procédure, etc.) sont généralement décrits de

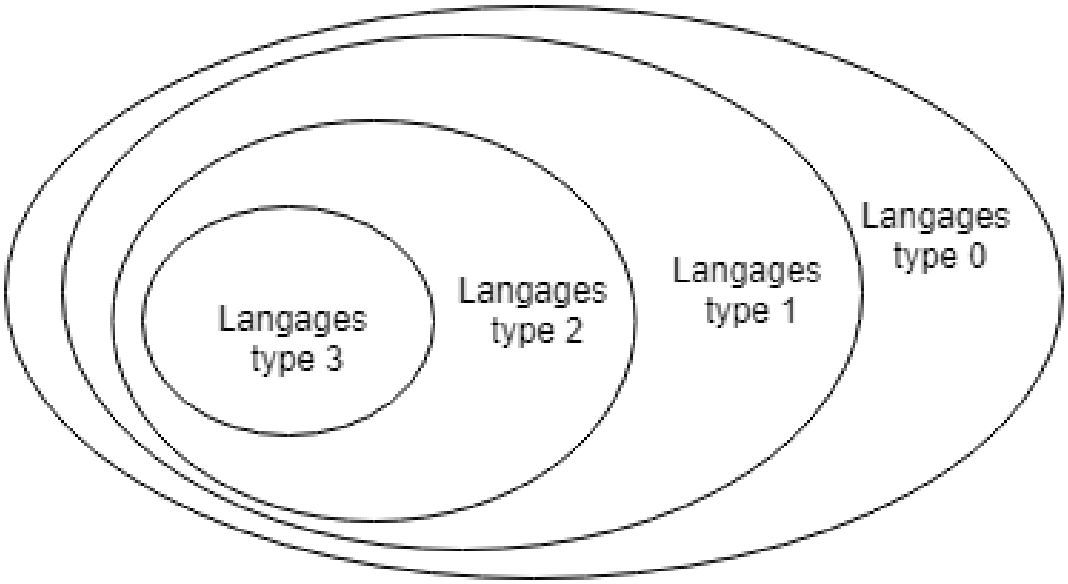

FIGURE 1.1 – Relation d'inclusion entre les types de langages.

manière informelle à coté de la grammaire, c'est ce qu'on appelle une analyse sémantique.

1.4 Propriétés de fermeture

Le miroir, la concaténation, l'union et l'étoile de Kleen sont appelées des opérations régulières.

Théorème : Les classes de langages de type i (i=0 ...i=3) sont fermées par rapport aux opérations régulières.
En d'autres termes, si L_1 et L_2 sont deux langages de type i alors :

- $(L_1)^R$ est de type i ;
- $L_1 \cup L_2$ est de type i ;
- $L_1.L_2$ est de type i ;
- L_1^* est de type i.

Chapitre 2

Langages réguliers et Automates à États Finis

Sommaire

Un automate à états finis est un outil fondamental de calcul et de modélisation en mathématique discrète, utilisé dans de nombreux domaines (Langages formels et compilation, protocoles de communication, contrôle de processus, etc.). C'est l'abstraction de tout système susceptible d'être dans un nombre fini d'états à des moments différents. Toutefois, dans ce cours, nous considérons qu'un AEF est une machine abstraite qui permet de lire un mot et d'affirmer son appartenance , ou pas, à un langage L.
La première partie de ce chapitre est consacrée aux AEF, leurs représentations et leurs variantes. Dans la seconde partie, nous étudions les expressions régulières et leurs propriétés.

2.1 Automate à états finis

Un automate à états finis est la plus simple machine abstraite (programme) qui reconnaît les langages réguliers (type3), il n'utilise aucune mémoire.

2.1.1 Définition

Un AEF est défini formellement par un quintuplé (X, Q, I, F, δ) où :

- X est un ensemble fini de symboles (les lettres de l'alphabet) ;
- Q est un ensemble fini d'états ;
- I est l'état initial, I$\in$Q ;
- F est l'ensemble des états finaux, F$\subset$Q ;
- δ est la fonction de transition, définie de Q$\times$X dans Q.

2.1.2 Représentations d'un AEF

Il existe trois principales représentations pour les AEF, à savoir la représentation matricielle, sous forme de graphe orienté étiqueté ou sous forme de fonction. Nous illustrons ces différentes représentation à travers l'exemple suivant. Soit A l'AEF défini par le quintuplé (X, Q, I, F, δ) tel-que :
X=$\{a, b\}$, Q=$\{q_0, q_1\}$, $I = q_0$, F=$\{q_0\}$ et on définit les transitions suivantes : $\delta(q_0, a) = q_0$, $\delta(q_0, b) = q_1$, $\delta(q_1, b) = q_1$.

2.1.2.1 Représentation matricielle

L'exemple précèdent nous donnera une matrice. La première colonne représente l'ensemble des états, la première ligne représente l'ensemble des lettres de l'alphabet. On remarque l'apparition du symbole ⇒ pour dire q_0 est l'état initial et du symbole ◎ pour dire que q_1 est un état final.

	δ	a	b
⇒	q_0	q_0	q_1
◎	q_1	-	q_1

Représentation matricielle

2.1.2.2 Représentation graphique

On représente un AEF par un graphe orienté étiqueté, les noeuds représentent les états, les arêtes représentent les transitions, les lettres de l'alphabet sont les étiquettes des arêtes, l'état initial est représenté par une flèche entrante, les états finaux sont représentés par un double cercle, cela est illustré dans la figure 2.1.

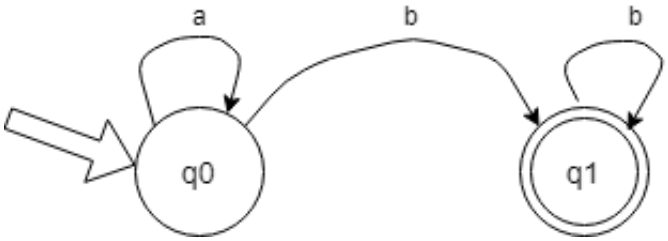

FIGURE 2.1 – Représentation d'un AEF sous forme de graphe orienté et étiqueté.

2.1.2.3 Représentation sous forme de relation

C'est une relation binaire entre deux ensembles, le premier contient toutes les combinaisons possibles de couples (état, lettre de l'alphabet), le second contient les états de l'automate, les flèches reliant des éléments du premier ensemble aux éléments du deuxième représentent les transitions. Notons que l'état initial et les états finaux doivent être définis sous forme d'ensembles, cela est illustré par la figure 2.2.
état initial= $\{q_0\}$ et F=$\{q_1\}$.

2.2 Langage reconnu par un AEF

Soit A=(X, Q, I, F, δ) un AEF.

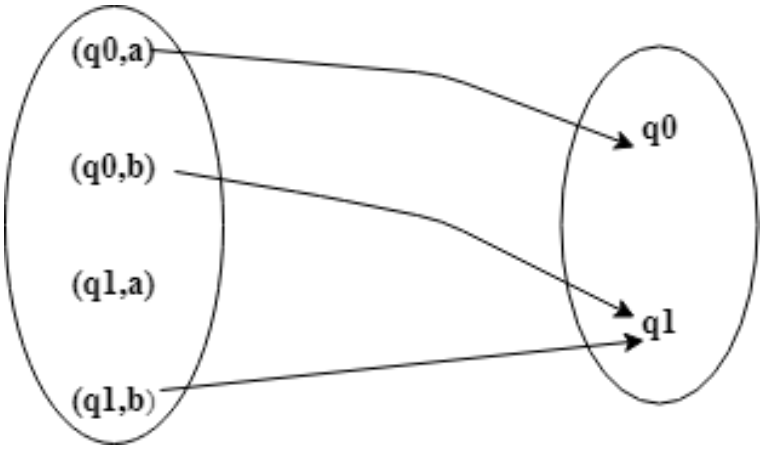

FIGURE 2.2 – Représentation d'un AEF sous forme de relation.

2.2.1 Définition d'une configuration

La configuration de l'AEF A, à un certain instant, est donnée par l'état courant de l'AEF et du mot qui reste à lire (état courant, mot qui reste à lire). La configuration initiale est (q_0, ω) où q_0 est l'état initial de l'AEF et ω le mot soumis à A (à reconnaître). La configuration finale est donnée par (q_f, ε) où q_f est un état final et le mot vide indique qu'il ne reste rien à lire (le mot appartient au langage).

2.2.2 Définition d'une dérivation directe

On dit qu'une configuration $(q_1,\ a\omega)$ dérive directement la configuration $(q_2,\ \omega)$ si et seulement si $\delta(q_1, a) = q_2$ où δ est la fonction de transition, a∈X et $\omega \in X^*$. On note $(q_1,\ a\omega) \models (q_2, \omega)$.

2.2.3 Définition d'une dérivation indirecte

On dit qu'une configuration $(q,\ \omega_1)$ dérive indirectement une autre configuration $(p,\ \omega_2)$, si et seulement s'il existe 0, 1 ou plusieurs dérivations directes qui, à partir de $(q,\ \omega_1)$, mènent à la configuration $(p,\ \omega_2)$. On note $(p,\ \omega_1) \models^* (q, \omega_2)$.

2.2.4 Définition du langage reconnu par un AEF

Le langage reconnu par l'AEF A noté $L(A)$ est defini par : $L(A) = \{\omega \in X^* / (q_0, \omega) \models^* (q_f, \varepsilon)\}$ avec $q_f \in F$.

Remarque : Un AEF reconnaît un et un seul langage, mais le même langage peut être reconnu par plusieurs AEF.

Interprétation : Un AEF est interprété comme une machine abstraite qui possède une bande (ruban) finie et divisée en cellules, sur lesquelles on insère le mot à reconnaître.

2.3 Équivalence des AEF

On dit que deux AEF A_1 et A_2 sont équivalents si et seulement s'ils reconnaissent le même langage, $L(A_1) = L(A_2)$.

Exemples Trouver des AEF qui reconnaissent les langages suivants :

- $L_1 = \{\omega \in \mathbb{N} / \, |\omega| \equiv 0\,[2]\}$
- $L_2 = \{\omega \in \mathbb{N} / \, |\omega| \equiv 1\,[5]\}$
- $L_3 = \left\{a^i b^j c^k / i, k \geq 1, j > 1\right\}$
- $L_4 = \left\{0^i b^2 1^k / i \geq 0, k > 1\right\}$
- $L_5 = \{\varepsilon\}$

Solutions : Nous proposons les solutions par représentation graphique comme cela apparaît sur les figures 2.3, 2.4, 2.5, 2.6et 2.7.

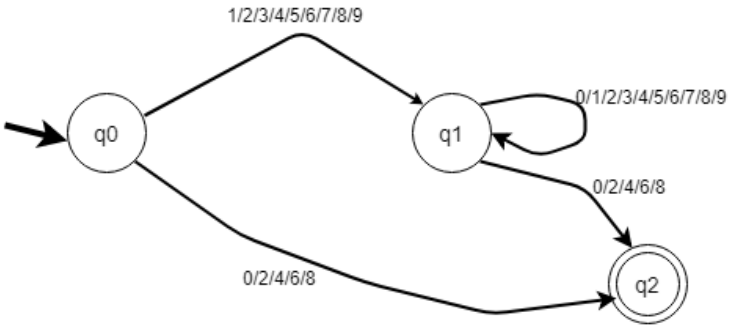

FIGURE 2.3 – AEF qui reconnaît le langage L_1.

2.4 Variantes d'AEF

Soit A=(X, Q, I, F, δ) un AEF.

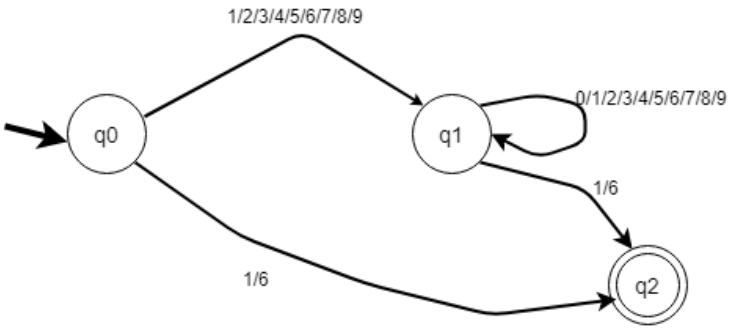

FIGURE 2.4 – AEF qui reconnaît le langage L_2.

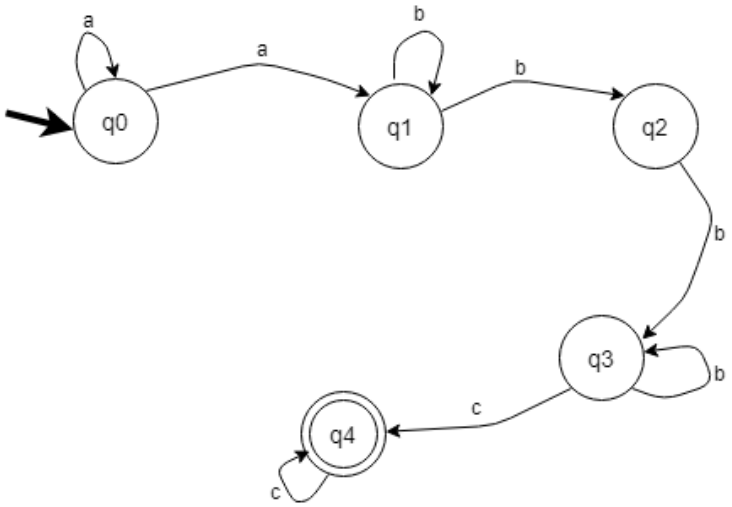

FIGURE 2.5 – AEF qui reconnaît le langage L_3.

2.4.1 AEF déterministe (AEFD)

A est dit déterministe si et seulement si la fonction de transition δ associe à chaque couple (q, a) au plus un état p, avec a∈X et p, q∈Q. Un exemple d'AEF déterministe est donné dans la figure 2.8.

2.4.2 AEF déterministe complet (AEFDC)

A est déterministe complet (complètement spécifié) si et seulement si à chaque paire (q, a)∈Q×X, la fonction δ associe exactement un état. Un exemple d'AEF déterministe complet est donné dans la figure 2.8.

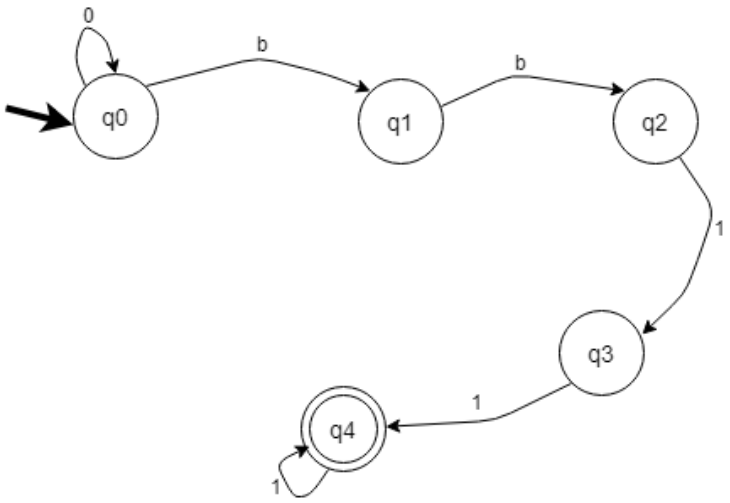

FIGURE 2.6 – AEF qui reconnaît le langage L_4.

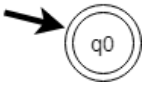

FIGURE 2.7 – AEF qui reconnaît le langage L_5.

2.4.3 AEF non déterministe (AEFND)

A est dit non déterministe si et seulement s'il existe au moins une pair (q, a)∈Q×X pour laquelle la fonction δ associe au moins deux états. Un exemple d'AEF non déterministe est donné dans la figure 2.8.

2.4.4 AEF généralisé (AEFG)

Dans un AEF généralisé les transitions (étiquettes) peuvent être engendrées par des mots. Les transitions causée par le mot ε sont appelées des transitions spontanées (ε-transitions), il s'agit d'un changement d'états sans lecture. Un exemple d'AEF généralisé est donné dans la figure 2.8.

2.4.5 AEF simple (AEFS)

Dans un AEF simple les transitions (étiquettes) sont toujours générées par un et un seul symbole de l'alphabet. Il y a trois AEF simples dans la figure 2.8.

2.4.6 AEF partiellement généralisé (AEFPG)

Dans un AEF partiellement généralisé les transitions (étiquettes) sont générées par un seul symbole de l'alphabet ou par ε. Un exemple d'AEF partiellement généralisé est donné dans la figure 2.8.

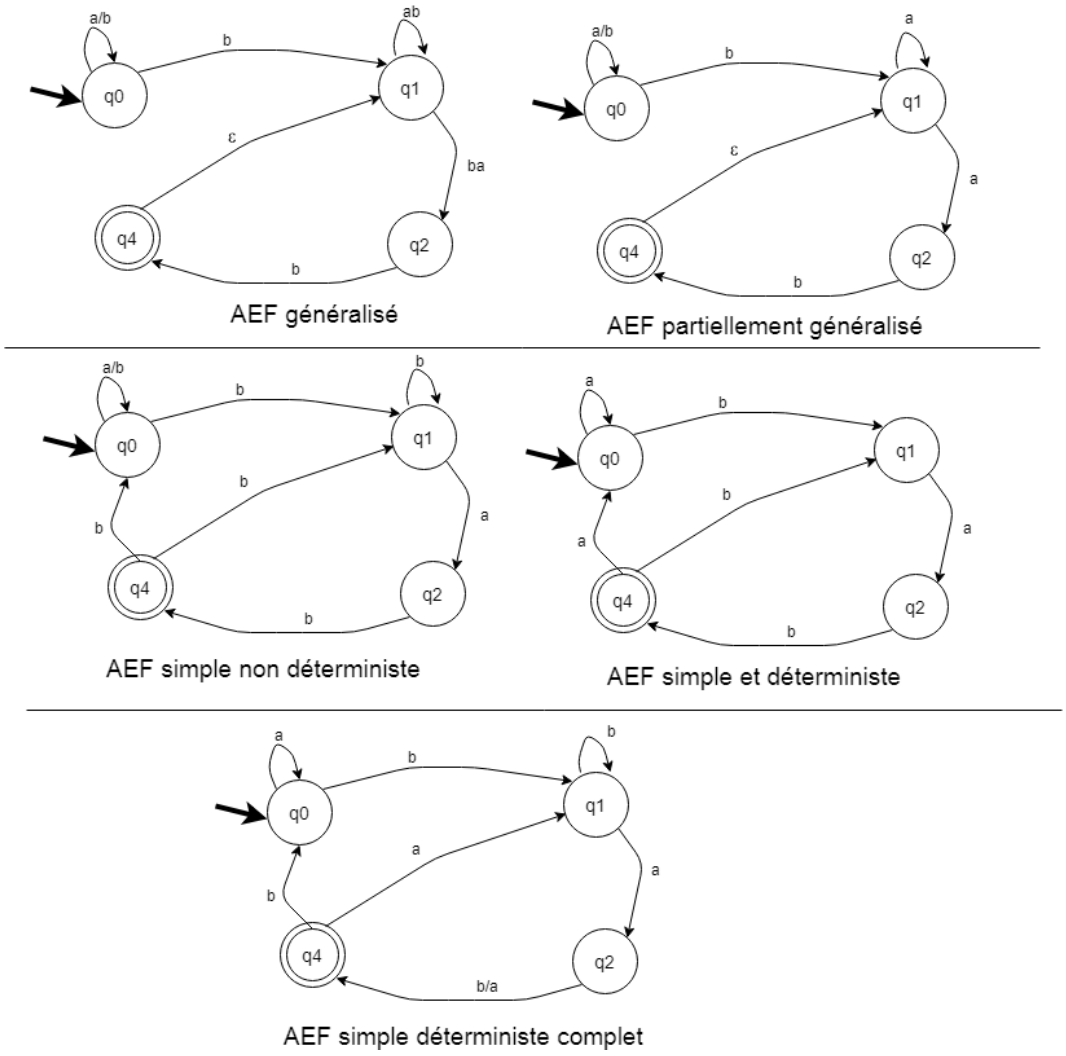

FIGURE 2.8 – Variantes d'AEF.

Théorème : Pour chaque AEF généralisé, il existe un AEF simple et déterministe qui reconnaît le même langage.

2.5 Transformation d'un AEF généralisé en AEF simple et déterministe

2.5.1 Élimination des mots contenant au moins deux lettres

Pour chaque transition avec un mot ω, tel-que $|\omega| = n$, avec $n \geq 2$, créer n-1 états supplémentaires et rajouter les transitions qui relient ces états avec les lettres de ω, à la fin de cette étape on obtient un AEF partiellement généralisé. Cela est illustré par la transformation de l'AEF généralisé sur la figure 2.9 en l'AEF partiellement généralisé sur la figue 2.10.

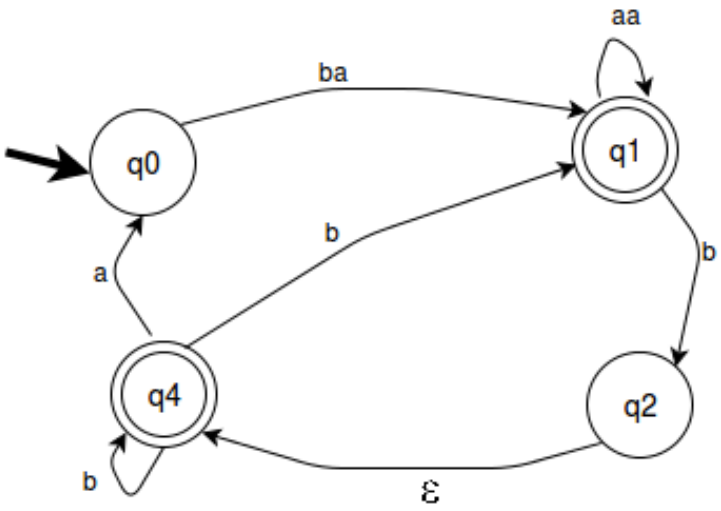

FIGURE 2.9 – AEF généralisé.

2.5.2 Élimination des ε-transitions

La suppression des ces transitions nous donne un AEF simple, pour ce faire il faut d'abord enlever les transitions par ε et :

$$\delta(q_i, \varepsilon) = q_j \begin{cases} Si \quad q_j \in F \quad alors \quad q_i \in F \\ \quad\quad et \\ \forall a \in X : \quad si \quad \delta(q_j, a) = q_k \quad alors \quad \delta(q_i, a) = q_k \end{cases}$$

Un exemple de cette transformation est illustré en transformant l'AEF partiellement généralisé de la figure 2.10 en l'AEF simple et non déterministe sur la figure 2.11.

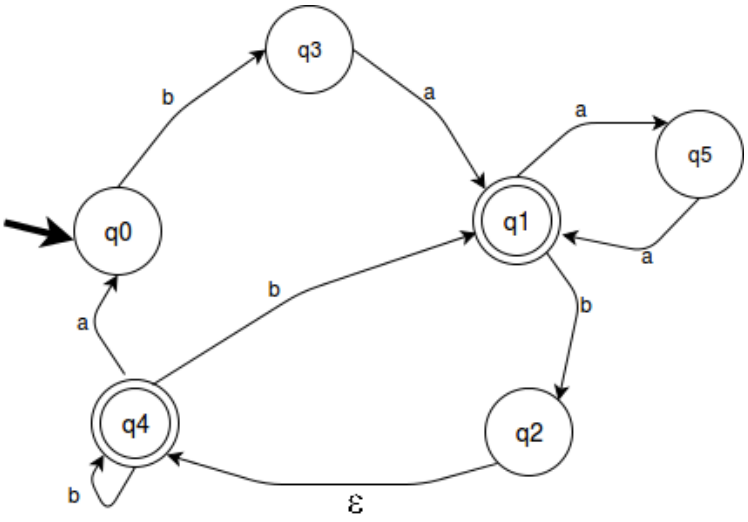

FIGURE 2.10 – AEF partiellement généralisé.

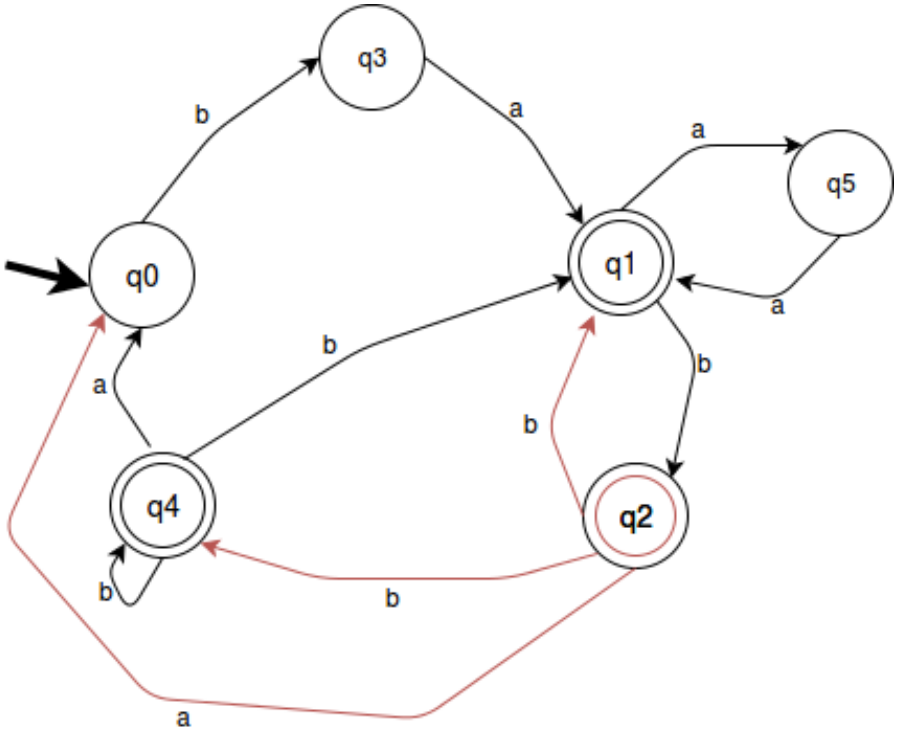

FIGURE 2.11 – AEF simple non déterministe.

2.5.3 Transformation d'un AEF non déterministe en un AEF déterministe

Dans certains ouvrages, avant d'effectuer cette transformation, on transforme d'abord l'AEF en AEF complet, ceci est surtout fait dans le cas ou nous voulons implémen-

ter notre AEF, ceci nous permettra d'avoir une matrice complètement définie, et nous n'aurons aucune case vide. En ce qui concerne notre cours, nous transformons directement notre AEF sans passer par cette étape. Pour transformer un AEF simple et non déterministe en AEF déterministe, il suffit de suivre les étapes suivantes :

- Partir de l'état initial $E^{(0)} = \{q_0\}$ (c'est l'état initial du nouvel automate) ;
- Construire $E^{(1)}$ l'ensemble des états obtenus à partir de $E^{(0)}$ par toutes les transitions avec les lettres de X. $E^{(1)} = E^{(0)} \cup \{q_i\} / \delta(q_0, x) = q_i, x \in X$ et $q_0 \in E^{(0)}$;
- Recommencer l'étape précédente pour toutes les transitions possibles et pour chaque nouvel ensemble $E^{(i)}$. $E^{(j)} = E^{(j-1)} \cup \{q_i\} / \delta(q_{i-1}, x) = q_i, x \in X$ et $q_{i-1} \in E^{(j-1)}$;
- Tous les ensembles contenant au moins un état final du premier automate deviennent des états finaux.

L'exécution de ces étapes sur l'AEF simple et non déterministe de la figure 2.11 est illustrée par la table des transitions suivantes :

		a	b
⇒	q0	/	q3
	q3	q1	/
◎	q1	q5	q2
	q5	q1	/
◎	q2	q0	q1q4
◎	q1q4	q0q5	q1q2q4
	q0q5	q1	q3
◎	q1q2q4	q0q5	q1q2q4

Table des transitions

Et nous obtenons enfin l'AEF simple et déterministe illustré par la figure 2.12.

2.6 Minimisation des AEF

Cette notion ne s'applique que dans le cas des AEF simples et déterministes.

2.6.0.1 Définition d'un AEF minimal (canonique)

Un AEF déterministe est minimal (canonique) si et seulement si tout autre AEF déterministe reconnaissant le même langage a au moins le même nombre d'états.

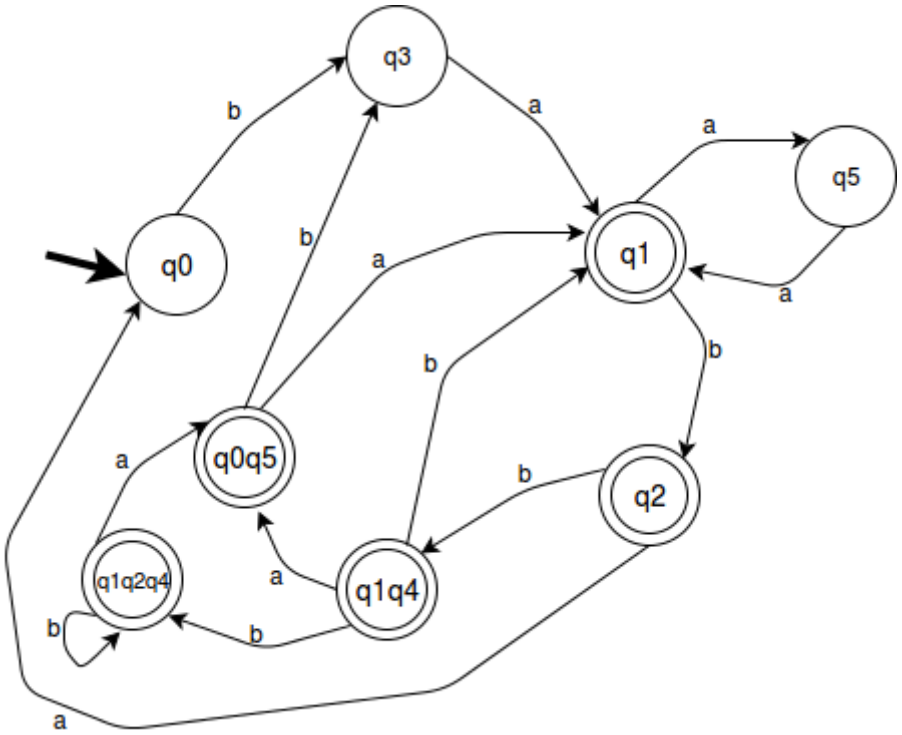

FIGURE 2.12 – AEF simple déterministe.

Proposition : Un AEF minimal est unique à la numérotation prés des états.

2.6.0.2 β-équivalence entre deux états

Soit A un AEF déterministe et complet, deux états q et p sont dits β-équivalents si et seulement s'ils permettent d'atteindre les états finaux en utilisant les mêmes mots. Dans l'AEF déterministe et complet de la figure 2.13, les états 5 et 6 ne peuvent pas être β-équivalents, car le 6 atteint un état final avec b mais pas le 5.

Remarque : Le nombre de classes d'équivalence de la relation β-équivalence est égal au nombre des états de l'AEF canonique, car les états de chaque classe d'équivalence reconnaissent le même langage (ils seront fusionnés).

Proposition : Si p et q sont des états d'un AEF A alors : pβq si et seulement si $\forall x \in X : \delta(p, x)\beta\delta(q, x)$.
Cette proposition est utilisée pour construire une suite de relations $(\beta_i)_{i\geq 0}$.

Construction de la relation β_i

1. β_0 contient deux classes, une pour les états finaux et une autre pour les états non finaux ;
2. si pβ_iq et $\forall x \in X : \delta(p,x)\beta_i\delta(q,x)$ alors p et q sont dans la même classe dans la relation β_{i+1} ;
3. Recommencer l'étape deux jusqu'à ce que $\beta_i = \beta_{i+1}$.

— L'ensemble des classes d'équivalence de la relation β_i finale sont les états de l'AEF minimal ;
— La classe qui contient l'ancien état initial devient l'état initial de l'automate minimal ;
— Toute classe contenant un état final devient un état final ;
— $\delta'(p',x) = q'$ si et seulement si $\exists q \in q', \delta(p,x) = q'$.

Exemple : Trouver l'AEF minimal de l'AEF représenté sur la figure 2.13.

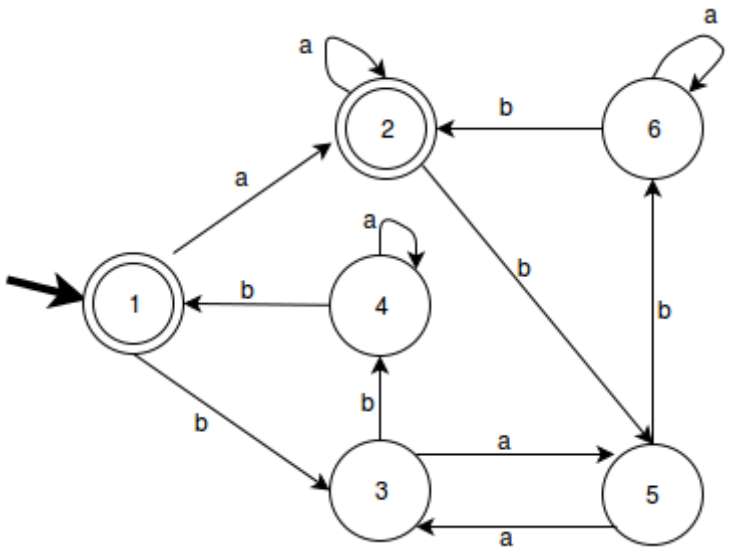

FIGURE 2.13 – AEF déterministe et complet.

Construction de la relation β_i

	1	2	3	4	5	6
B_0	I	I	II	II	II	II
a	I	I	II	II	II	II
b	II	II	II	I	II	I
B_1	I	I	II	III	II	III
a	I	I	II	III	II	III
b	II	II	III	I	III	I
B_2	I	I	II	III	II	III

$I = \{1, 2\}, II = \{3, 5\}, III = \{4, 6\}$, L'AEF minimal est représenté sur la figure 2.14.

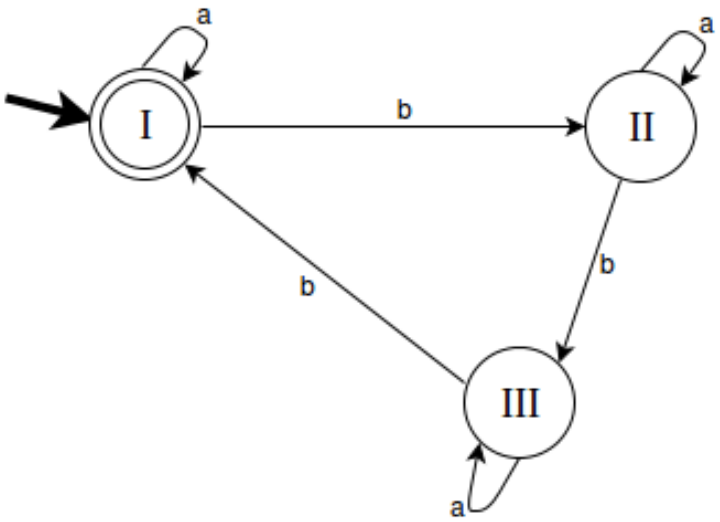

FIGURE 2.14 – AEF minimal.

2.7 Langages réguliers et AEF

2.7.1 Rappel de la définition d'un langage régulier

Un langage est dit régulier (rationnel) s'il existe une grammaire régulière à gauche ou régulière à droite qui le génère.

Définition d'une grammaire régulière à droite : Soit la grammaire G=(T, N, S, P), G est dite régulière à droite si et seulement si toutes ses règles de production ont l'une des formes suivantes : $A \to \omega B$ ou $A \to \omega$ avec A, B$\in$N et $\omega \in T^*$.

Définition d'une grammaire régulière à gauche : Soit la grammaire G=(T, N, S, P), G est dite régulière à gauche si et seulement si toutes ses règles de production ont l'une des formes suivantes : $A \to B\omega$ ou $A \to \omega$ avec A, B∈N et $\omega \in T^*$.

2.7.1.1 Langages réguliers et AEF

Proposition 1 : Pour chaque langage régulier L, il existe un AEF A tel que L(A)=L.

Proposition 2 : Le langage reconnu par un AEF est un langage régulier.

2.7.2 Grammaires et AEF

Proposition 1 : Pour toute grammaire régulière à droite $G = (T, N, S, P)$, il existe un AEF généralisé $A = (X, Q, q_0, \delta, F)$ tel que L(G)=L(A). Soit $G = (T, N, S, P)$ une grammaire régulière à droite, il s'agit de trouver un AEF $A = (X, Q, q_0, \delta, F)$ tel que L(G)=L(A). $X = T$. $Q = N \cup q_f$ tel que $q_f \in F$. $q_0 = S$. $F = \{q_f\}$. Pour chaque règle $A \to \omega B$, on associe la transition $\delta(A, \omega) = B$.

Pour chaque règle de la forme $A \to \omega$, on associe la transition $\delta(A, \omega) = q_f$.

Exemple : Trouvons l'AEF généralisé de la grammaire régulière à droite G qui possède les règles de production suivantes :
$S \to aaS \mid bA$
$A \to bA \mid cB \mid c$
$B \to cB \mid c$
$X = \{a, b, c\}$, $Q = \{S, A, B, q_f\}$, l'état initial est S, et l'ensemble des états finaux et $F = \{q_f\}$, les transitions de l'AEF sont définies dans la figure 2.15.

Proposition 2 : Pour tout AEF généralisé $A = (X, Q, q_0, \delta, F)$, il existe une grammaire régulière à droite $G = (T, N, S, P)$ tel que L(G)=L(A).
Soit l'AEF $A = (X, Q, q_0, \delta, F)$, il s'agit de trouver une grammaire régulière a droite $G = (T, N, S, P)$ tel que L(A)=L(G).
$T = X$;
$N = Q$;
$S = q_0$;
Si $\delta(p, \omega) = q$, alors $p \to \omega q \in P$;

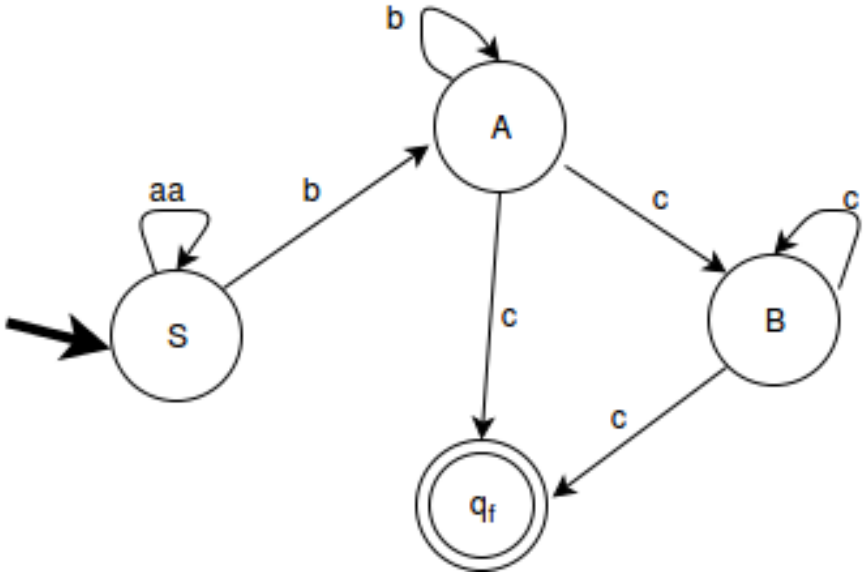

FIGURE 2.15 – AEF qui reconnaît le langage généré par la grammaire G.

Si $\delta(p, \omega) = q_f$, alors $p \to \omega \in P$;
Si $q_0 \in F$, alors $q_0 \to \varepsilon \in P$.

Exemple : Trouvons la grammaire G qui génère le langage reconnu par l'AEF généralisé de la figure 2.10.
$T = \{a, b\}$, $N = \{q_0, q_1, q_2, q_4\}$, l'axiome de la grammaire est q_0 et les règles de production sont les suivantes :
$q_0 \to baq_1 \mid ba$
$q_1 \to aaq_1 \mid bq_2 \mid aa$
$q_2 \to q_4 \mid \varepsilon$
$q_4 \to bq_4 \mid bq_1 \mid aq_0 \mid b$

2.7.3 Propriétés de fermeture de la classe des langages réguliers

En plus des opérations régulières (. , *, l'union et le miroir), la classe des langages réguliers est fermée par rapport au complément et à l'intersection.

2.7.4 Expressions régulières

Soit X un alphabet, les expressions régulières (rationnelles) sur l'alphabet X sont définies par récurrence comme suit :

Cas de base :

— $\varnothing$ est une expression régulière qui dénote (représente) le langage vide ;
— ε est une expression régulière qui dénote le langage $\{\varepsilon\}$;
— a (où a∈X) est une expression régulière qui dénote le langage $\{a\}$.

Induction : Si r et s sont deux expressions régulières qui dénotent respectivement les langages R et S alors :

— $r + s$ est l'expression régulière qui dénote $R \cup S$;
— $r.s$ est l'expression régulière qui dénote $R.S$;
— r^* est l'expression régulière qui dénote R^* ;
— r^+ est l'expression régulière qui dénote R^+.

Convention : L'étoile de Kleen et le plus en exposant sont plus prioritaires que la concaténation, qui est elle même plus prioritaire que le plus sur la ligne.

Proposition : Deux expressions régulières sont dite ε-équivalentes si et seulement si elle dénotent le même langage.

2.7.4.1 Propriétés sur les expressions régulières

Si P, Q et R sont trois expressions régulières alors :

- $P + Q = Q + P$;
- $(P + Q) + R = P + (Q + R)$;
- $(P.Q).R = P.(Q.R)$;
- $P.\varnothing = \varnothing.P = \varnothing$;
- $P.\varepsilon = \varepsilon.P = P$;
- $P + \varnothing = \varnothing + P = P$;
- $P^*.P = P.P^*$;
- $(P + \varepsilon)^*.R = P^*.R$;
- $P^*.P^* = P^* = P^+$;
- $P^* = \varepsilon + P.P^*$;

- $\varnothing^* = \varepsilon$;
- $(P^*)^* = P^*$;
- $(P^* + Q^*)^* = (P + Q)^* = (P^*.Q^*)^*$.

Proposition : Pour toute expression régulière E, il existe un AEF qui reconnaît le langage dénoté par E.

Proposition : Pour tout AEF A, il existe une expression régulière E qui dénote le langage reconnu par A.

Exemples : Les expressions régulières qui dénotent les langages reconnus par les AEF A_1 et A_2 illustrés sur la figure 2.16sont :
$E_1 = (a + b)^* b(abb)^* a$
$E_2 = b(a^*(abb)^* + a^*(abab)^*)^*(\varepsilon + ab) = b(a^*(abb)^*(abab)^*)^*(\varepsilon + ab) = b(a + abb + abab)^*(\varepsilon + ab)$

2.7.5 Dérivées

Soit L un langage sur un alphabet X et $\omega \in X^*$, la dérivée de L par rapport à ω, notée $L \parallel \omega$, est définie par : $L \parallel \omega = \{z \in X^* / \omega.z \in L\}$.

Exemples : Soient les langages $L_1 = \{\varepsilon, a, ab, aa, ba\}$ et $L_2 = \{a^n / n \geqslant 0\}$.
$L_1 \parallel a = \{\varepsilon, b, a\}$;
$L_1 \parallel aa = \{\varepsilon\}$;
$L_2 \parallel a = L_2$.

2.7.5.1 Propriétés des dérivées

- $a_i \parallel a_j = \begin{cases} \varepsilon & si \quad a_i = a_j \\ \varnothing & sinon \end{cases}$;
- $(\bigcup L_i) \parallel a = \bigcup (L_i \parallel a)$;
- $(L_1.L_2) \parallel a = \begin{cases} (L_1 \parallel a).L_2 & si \quad \varepsilon \notin L_1 \\ (L_1 \parallel a).L_2 \cup L_2 \parallel a & sinon \end{cases}$;
- $L^* \parallel a = (L \parallel a).L^*$;
- $L \parallel \omega_1.\omega_2 = (L \parallel \omega_1) \parallel \omega_2$;

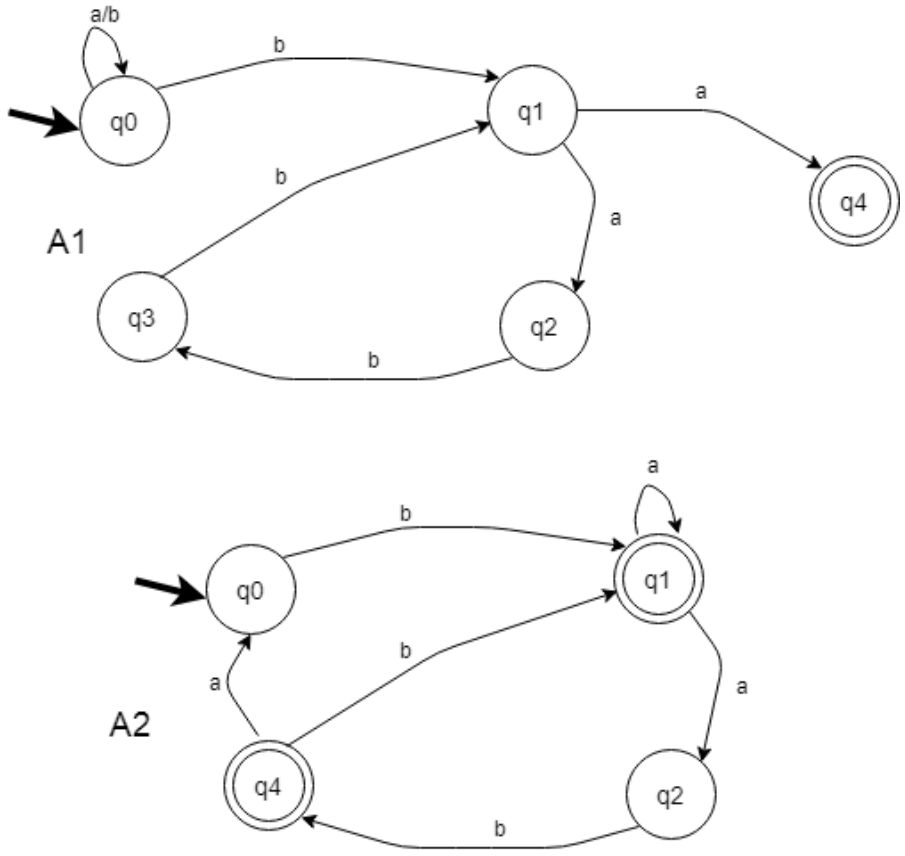

FIGURE 2.16 – De l'AEF vers une l'expression régulière.

- $L \parallel \omega = \varnothing$ si aucun mot de L ne commence par ω ;
- $\varepsilon \parallel a = \varnothing$;
- $\omega.L \parallel \omega = L$.

Exemple : Soit le langage $L = (a+b)^*ab(bb+a)^*$. Calculer $L \parallel a$.
$L \parallel a = (a+b)^* \parallel a.ab(bb+a)^* + ab(bb+a)^* \parallel a$
$=(a+b) \parallel a.(a+b)^*ab(bb+a)^* + b(bb+a)^*$
$=(a \parallel a + b \parallel a).a.(a+b)^*ab(bb+a)^* + b(bb+a)^*$
$=(a+b)^*ab(bb+a)^* + b(bb+a)^*$.

2.7.6 Critères de régularité d'un langage

2.7.6.1 Théorème de Nerode

Si L un langage sur un alphabet X, L est régulier si et seulement si le nombre de ses dérivées par rapport aux mots de x^* est fini.

Méthodes pour montrer qu'un langage est régulier : On peut montrer la régularité d'un langage L, par l'une des méthodes suivantes :

- Tous les langages finis sont réguliers ;
- Si on trouve un AEF qui reconnaît un langage L, alors L est régulier ;
- Si on trouve une grammaire régulière générant L, alors ce langage est régulier ;
- On peut utiliser le théorème de Nerode pour montrer qu'un langage est régulier ;
- On peut exploiter les propriétés de fermeture pour montrer qu'un langage est régulier.

Exemple : En utilisant le théorème de Nerode, montrer que $L = \{a^i b^j, i, j \geqslant 1\}$ est régulier.
On calcule les dérivées du langage par rapport aux lettres de l'alphabet, jusqu'à ce qu'on tombe sur les mêmes ensembles.
$\{a^i b^j, i, j \geqslant 1\} \| \text{a} = \{a^i b^j, i \geqslant 0, j \geqslant 1\} = L_1$
$\{a^i b^j, i, j \geqslant 1\} \| \text{b} = \{b^j, j \geqslant 0\} = L_2$
$\{a^i b^j, i \geqslant 0, j \geqslant 1\} \parallel a = \{a^i b^j, i \geqslant 0, j \geqslant 1\} = L_1$
$\{a^i b^j, i \geqslant 0, j \geqslant 1\} \parallel b = \{b^j, j \geq 0\} = L_2$
$\{b^j, j \geqslant 0\} \parallel a = \varnothing$
$\{b^j, j \geqslant 0\} \parallel b = \{b^j, j \geqslant 0\} = L_2$
Le nombre de dérivées est fini, donc le langage L est régulier.

Remarque : On peut construire l'AEF reconnaissant L à partir des dérivées et des ensembles trouvés. Dans l'exemple précèdent, on peut déduire que l'AEF contient trois états et cinq transitions, les états correspondent aux ensembles trouvés au cours du calcul et les transitions correspondent aux dérivations.

$L_2 = \{a^i b^i c^j, i, j \geqslant 0\} = L_1.\{c^j, j \geq 0\}$, Or le langage L_1 n'est pas régulier, et puisque la classe des langages réguliers est fermée par rapport à la concaténation, alors L_2 n'est pas régulier.

Chapitre 3

Langages algébriques et Automates à Piles (AàP)

Sommaire

Les langages de type 2, appelés aussi les langages algébriques ou encore les langage hors contexte, sont reconnus par des machines abstraites semblables aux AEF à une différence prêt qui est la mémoire, cette mémoire est une pile. Le présent chapitre est consacré à ce type de langages ainsi qu'aux automates à piles.

3.1 Rappel sur les grammaires et langages algébriques

Définition d'une grammaire de type 2 : Une grammaire G=(T, N, S, P) est dite de à contexte libre (Algébrique ou de type 2) si et seulement si toutes ses règles de production sont sous la forme : $A \to B$ avec $A \in$ N et $B \in (T \cup N)^*$.

Définition des langages de type2 (hors contexte ou algébrique) : Ce sont les langages qui peuvent être définis par des grammaires de type 2.

Remarque : L'ensemble des langages réguliers est inclus dans l'ensemble des langages algébriques.

3.2 Arbre syntaxique

Vue l'utilisation d'un seul symbole non terminal à gauche des règles de production dans les grammaires à contexte libre, il est toujours possible de construire un arbre de dérivation pour un mot généré.

Définition Soit la grammaire G=(T, N, S, P) et soit $\omega \in L(G)$. Un arbre syntaxique associé à ω est construit tel-que :

- La racine de l'arbre est étiquetée par l'axiome ;
- Les noeuds intermédiaires contiennent des non terminaux ;
- Les feuilles sont des terminaux.

La lecture de gauche à droite des feuilles de l'arbre reconstitue le mot auquel l'arbre est associé.

3.2.1 Notion d'ambiguïté

Définition d'un mot ambigu : Un mot ω est dit ambigu si et seulement s'il existe deux arbres de dérivation différents qui lui sont associés.

Définition d'une grammaire ambiguë : Une grammaire G est dite ambiguë si et seulement s'il existe au moins un mot ambigu appartenant à L(G).

Exemple : Soit G la grammaire qui possède les règles de production suivantes :
S→S∧S|S∨S|S⇒S|S⇔S|¬S |q|p

Question : Montrer que G est ambiguë.

Réponse : Le mot p∧q ⇔p est ambigu, car il existe deux dérivations différentes qui nous permettent de l'atteindre, si nous numérotons les règles de 1 à 7 alors on aura :
$S \vdash^{(4)} S \Leftrightarrow S \vdash^{(1)} S \wedge S \Leftrightarrow S \vdash^{(7)} p \wedge S \Leftrightarrow S \vdash^{(6)} p \wedge q \Leftrightarrow S \vdash^{(7)} p \wedge q \Leftrightarrow p.$
$S \vdash^{(1)} S \wedge S \vdash^{(6)} p \wedge S \vdash^{(4)} p \wedge S \Leftrightarrow S \vdash^{(6)} p \wedge q \Leftrightarrow S \vdash^{(7)} p \wedge q \Leftrightarrow p.$
Par conséquent la grammaire G est ambiguë.

Remarque 1 : Certains langages peuvent être générés, à la fois, par des grammaires ambiguës et des grammaires non ambiguës.

Remarque 2 : Il n'existe aucun algorithme qui permet de trouver une grammaire non ambiguë (si elle existe) qui génère un langage.

3.3 Forme normale de Chomsky

Une grammaire G=(T, N, S, P) est dite sous forme normale de Chomsky (FNC) si et seulement si toutes ses règles de production sont sous la forme $A \to BC$ où $A \to a$ avec $A, B, C \in$ N et $a \in T$.

Proposition : Pour toute grammaire algébrique, il existe une grammaire équivalente sous forme normale de Chomsky.

L'intérêt pratique de la FNC est que les arbres de dérivations sont des arbres binaires, ce qui facilite l'application des algorithmes d'exploration des arbres.

3.3.1 Conversion d'une grammaire algébrique en FNC

Pour obtenir une grammaire sous forme normale de Chomsky équivalente à une grammaire algébrique G, il faut :

1. Transformer la grammaire en une grammaire propre ;
2. Pour chaque terminal a, introduire le non terminal C_a, puis rajouter la règle $C_a \to a$;
3. Pour chaque règle $A \to \alpha$, avec $|\alpha| \geq 2$, on remplace chaque terminal par le non terminal qui lui est associé ;
4. Pour chaque règle $A \to \beta$, avec $|\beta| \geq 3$, $(\beta = \beta_1\beta_2...\beta_n)$, créer les non terminaux D_i, puis remplacer la règle concernée par les règles suivantes : $A \to \beta_1 D_1$, $D_1 \to \beta_2 D_2, ...D_{n-2} \to B_{n-1}B_n$. où $D_i = B_{i+1}B_{i+2}...B_n$, avec i variant de 1 jusqu'à $n-2$.

3.4 Forme normale de Greibach

Une grammaire algébrique est sous forme normale de Greibach (FNG) si et seulement si toutes ses règles de production sont sous la forme $A \to x\alpha$ ou $S \to \varepsilon$, avec $x \in T$, $\alpha \in N^*$ et S est l'axiome.

Proposition : Pour toute grammaire algébrique G_1, il existe une grammaire G_2 sous forme normale de Greibach tel-que $L(G_2) = L(G_1)$.

L'intérêt pratique de la FNG est qu'à chaque dérivation, on détermine un préfixe de plus en plus long formé uniquement de symboles terminaux. Cela permet de construire des automates à piles à partir des grammaires plus aisément, et par conséquent des analyseurs syntaxiques sont facilement implémentables.

3.5 Automates à piles

Comme les AEF, les automates à piles (AàP) sont des machines abstraites qui affirment, ou pas, l'appartenance d'un mot à un langage. Les langages reconnus par les AàP sont les langages algébriques (Type 2). En plus des composantes des AEF, les AàP possèdent une pile pour le stockage.

Exemple préliminaire : Les étapes de reconnaissance du langage $\{a^i b^i, i \geq 1\}$ par un AàP pourraient être les suivantes :

1. Lire les a, les stocker dans la pile et ne pas changer d'état ;

2. A la rencontre du premier b, dépiler un a et changer d'état ;
3. Dépiler un a pour chaque b rencontré ;
4. Si les a de la pile se terminent au même moment que les b lus, alors le mot appartient au langage.

3.5.1 Définition formelle

Un AàP est défini formellement par un septuplé $(\Sigma, \Gamma, Q, q_0, \delta, Z_0, F)$ où :

- Σ est l'alphabet d'entrée ;
- Γ est l'ensemble des symboles auxiliaires. $\Gamma \cup \Sigma$ est l'alphabet de la pile ;
- Q est un ensemble fini d'états ;
- q_0 est l'état initial ;
- Z_0 est le symbole de fond de pile, $Z_0 \in \Gamma$;
- F est l'ensemble des états finaux (d'acceptation) ;
- δ est la fonction de transition, elle est définie de $(\Gamma \cup \Sigma) \times Q \times \Sigma$ dans $(\Gamma \cup \Sigma)^* \times Q$.

$\delta(u, q, a) = (\alpha, p)$ peut avoir trois effets sur la pile, selon la valeur de α :

- Si $\alpha = u$, alors pas de changement dans la pile ;
- Si $\alpha = \varepsilon$, cela signifie qu'on a dépilé u ;
- $\alpha = uv_1v_2...v_n$ avec $v_i \in \Gamma^*$, cela signifie qu'on empile v_1, puis v_2 ... puis v_n dans cet ordre.

Exemple : Trouver l'AàP qui reconnait le langage $L = \{a^i b^i, i \geq 1\}$.

Réponse :

- $\delta(z_0, q_0, a) = (z_0a, q_1)$ (empiler le premier a et et passer à l'état q_1) ;
- $\delta(a, q_1, a) = (aa, q_1)$ (empiler tous les autres a et rester l'état q_1) ;
- $\delta(a, q_1, b) = (\varepsilon, q_2)$ (à la rencontre du premier b, dépiler un a et passer à l'état q_2) ;
- $\delta(a, q_2, b) = (\varepsilon, q_2)$ (dépiler un a pour chaque b et rester dans l'état q_2) ;
- $\delta(z_0, q_2, \varepsilon) = (z_0, q_{final})$ (si la pile se vide au même moment que le mot à reconnaître se termine, alors passer à un état final).

Notation : Pour des raisons de légèreté d'écriture, au lieu d'écrire $\delta(z_0, q_0, a) = (z_0a, q_1)$, nous écrivons $z_0q_0a \rightarrow z_0aq_1$.

3.5.2 Langage reconnu par un AàP

Soit A=$(\Sigma, \Gamma, Q, q_0, \delta, Z_0, F)$ un AàP.

3.5.2.1 Définition d'une configuration

Une configuration de l'AàP A, à un certain instant, est donnée par le contenu de la pile, l'état courant de l'AàP et du mot qui reste à lire (contenu de la pile, état courant, mot qui reste à lire). La configuration initiale est (Z_0, q_0, ω), où q_0 est l'état initial de l'AàP et ω le mot soumis à A (à reconnaître).

3.5.3 Définition d'une dérivation directe

On dit qu'une configuration $(\alpha u, q, a\omega)$ dérive directement la configuration $(\alpha\beta, p, \omega)$ si et seulement si $uqa \rightarrow \beta p$. On note $(\alpha u, q, a\omega) \models (\alpha\beta, p, \omega)$.

3.5.4 Définition d'une dérivation indirecte

On dit qu'une configuration (α, q, ω_1) dérive indirectement une autre configuration (β, p, ω_2), si et seulement s'il existe 0, 1 ou plusieurs dérivations directes qui, à partir de (α, q, ω_1), mènent à la configuration (β, p, ω_2). On note $(\alpha, q, \omega_1) \models^* (\beta, p, \omega_2)$.

3.5.5 Définition du langage reconnu par un AàP

Les AàP peuvent reconnaître les langages de deux modes differents, à savoir la reconnaissance par état final et la reconnaissance par pile vide.

3.5.5.1 Reconnaissance par état final

Le langage reconnu par l'AàP A par état final, noté $L(A)$, est défini par : $L(A) = \{\omega \in X^*/(z_0, q_0, \omega) \models^* (\alpha, q_f, \varepsilon)\}$ avec $q_f \in F$.

3.5.5.2 Reconnaissance par pile vide

La notion d'état final dans ce type de reconnaissance n'existe pas, ainsi un AàP qui reconnaît par pile vide est défini par un sextuplé $(\Sigma, \Gamma, Q, q_0, \delta, Z_0)$ (et non un septuplé). ainsi le langage reconnu par pile vide par un automate A est défini par $L(A) = \{\omega \in X^*/(z_0, q_0, \omega) \models^* (\varepsilon, q, \varepsilon)\}$.

3.5.5.3 Proposition :

Le mode de reconnaissance par état final est équivalent au mode de reconnaissance par pile vide.

Remarque : Un AàP reconnaît un et un seul langage, mais le même langage peut être reconnu par plusieurs AàP.

3.6 Équivalence des AàP

On dit que deux AàP A_1 et A_2 sont équivalents si et seulement s'ils reconnaissent le même langage, $L(A_1) = L(A_2)$.

Remarque : Il existe des variantes d'AàP ou on ne peut empiler plus d'un caractère à la fois, et d'autres qui permettent l'empilement de plusieurs caractères. les deux variantes d'automates sont équivalentes.

Exercice : I) Trouver des AàP qui reconnaissent les langages suivants :
$L_1 = \{a^i b^j / i, j \geq 1\}$;
$L_2 = \{a^i b^j / i \geq j \geq 1\}$;
$L_3 = \{a^i b^j / j \geq i \geq 0\}$.
II) Soit l'AàP A=$(\Sigma, \Gamma, Q, q_0, \delta, Z_0)$ suivant :
$\Sigma = \{a, b\}$;
$\Gamma = \{z_0, 1\}$;
$Q = \{q_0, q_1\}$;
q_0 est l'état initial ;
Z_0 Symbole initial de la pile.
Et nous définissons la fonction de transition par les instructions suivantes :

1. $z_0 q_0 a \rightarrow z_0 a q_0$
2. $a q_0 a \rightarrow a a q_0$
3. $a q_0 b \rightarrow q_1$
4. $a q_1 b \rightarrow q_1$
5. $z_0 q_1 b \rightarrow z_0 q_2$
6. $z_0 q_2 b \rightarrow z_0 q_2$

7. $z_0q_2 \rightarrow q_2$
8. $aq_1 \rightarrow q_3$
9. $aq_3 \rightarrow q_3$
10. $z_0q_3 \rightarrow q_3$

Questions :

1. Les mots ab, abb, aab appartiennent-ils à L(A) ?
2. Trouver le langage L(A).

Solution Partie I :

L'AàP du langage L_1 : L_1 est un langage régulier, on peut donc trouver un AàP sans l'exploitation de la pile. Soient les instructions suivantes :

1. $z_0q_0a \rightarrow z_0q_1$, pour lire le premier a (au moins un a).
2. $z_0q_1a \rightarrow z_0q_1$, pour lire tous les autres a.
3. $z_0q_1b \rightarrow z_0q_2$, pour lire le premier b (au moins un b).
4. $z_0q_2b \rightarrow z_0q_2$, pour lire tous les autres b.
5. $z_0q_2 \rightarrow z_0q_{final}$, une fois qu'il n y a plus de b à lire, l'AàP transite vers un état final.

L'AàP du langage L_2 :

1. $z_0q_0a \rightarrow z_0aq_0$
2. $aq_0a \rightarrow aaq_0$
3. $aq_0b \rightarrow q_1$
4. $aq_1b \rightarrow q_1$
5. $aq_1 \rightarrow q_{final}(i > j)$
6. $z_0q_1 \rightarrow q_{final}(i = j)$

L'AàP du langage L_3 :

1. $z_0q_0a \rightarrow z_0aq_0$
2. $aq_0a \rightarrow aaq_0$

3. $aq_0b \to q_1$
4. $aq_1b \to q_1$
5. $z_0q_1b \to z_0q_2$
6. $z_0q_2b \to z_0q_2$
7. $z_0q_2 \to z_0q_{final}$
8. $z_0q_0 \to z_0q_final$, Pour reconnaître ε.
9. $z_0q_b \to z_0q_2$, Dans le cas ou il n y a pas de a.
10. $z_0q_1 \to z_0q_{final}$Pour reconnaître le mot ab.

Solution Partie II : Dans ce qui suit, nous étudions l'appartenance des mots :
L'appartenance du mot ab :

Pile	état	mot	inst
z_0	q_0	ab	1
z_0a	q_0	ab	3
z_0	q_1	ab	aucune instruction donc ab $\notin$ L(G)

L'appartenance du mot abb :

pile	état	mot	instruction
z_0	q_0	abb	1
z_0a	q_0	abb	3
z_0	q_1	abb	5
z_0	q_2	abb	7
/	q_2	abb	pile vide et rien à lire, donc abb $\in$ L(G)

L'appartenance du mot aab :

pile	état	mot	instruction
z_0	q_0	aab	1
z_0a	q_0	aab	2
$z_0$aa	q_0	aab	3
z_0a	q_1	aab	8
z_0	q_3	aab	10
/	q_3	aab	pile vide et rien à lire, donc aab $\in$ L(G)

Langage reconnu par l'AàP : $L(A) = \{a^ib^j / i \neq j, i \geqslant 1, j \geqslant 1\}$.

3.7 AàP déterministe et non déterministe

Il existe deux cas de non déterminisme pour les AàP :

1. Pour le même sommet de pile, même état et le même symbole d'entrée, il existe au moins deux transitions ;
2. Pour le même sommet de pile et même état, on a le choix de lire ou ne pas lire du ruban.

Définition : Un AàP A=$(\Sigma, \Gamma, Q, q_0, \delta, Z_0, F)$ est dit déterministe si et seulement si pour chaque triplé (u, q, a) défini dans $\Sigma \times Q \times \Gamma$, la fonction δ associe au plus une paire (α, p), et si $\delta(u, q, a)$ est définie, alors il n'existe pas de transition $\delta(u, q, \varepsilon)$.

Remarque : Il existe des langages algébriques pour lesquels il n'existe pas d'AàP déterministe les reconnaissant.

Théorème : Si un Langage L est reconnu par un automate à pile déterministe, alors il existe une grammaire algébrique non ambiguë générant L.

3.8 Langages Algébriques et AàP

Proposition 1 : Pour chaque langage algébrique L, il existe un AàP A tel-que L(A)=L.

Proposition 2 : Les langages reconnus par des AàP sont des langages algébriques.

3.9 Grammaires et AàP

Proposition : Pour toute grammaire algébrique $G = (T, N, S, P)$, il existe un AàP A=$(\Sigma, \Gamma, Q, q_0, \delta, Z_0, F)$ tel-que L(G)=L(A).

Soit la grammaire algébrique G=(T, N, S, P), il s'agit de trouver un AàP A=$(\Sigma, \Gamma, Q, q_0, \delta, Z_0, F)$ reconnaissant le langage L(G) par pile vide.

1. En premier lieu, on transforme G sous forme normale de Greibach.

2. Ensuite, on définit les paramètres de l'AàP comme suit :
   $\Sigma = T$;
   $\Gamma = N$;
   $Q = \{q_0\}$;
   $z_0 = S$;
   Définition de δ : $\delta(x, q_0, x) = (\varepsilon, q_0)$ et si $A \to \alpha \in P$ alors $\delta(A, q_0, \varepsilon) = (\alpha^R, q_0)$.

Exemple : Soit G la grammaire qui est définie par les règles de production suivantes :
$S \to aA \mid bB \mid \varepsilon$
$A \to bS \mid aAA$
$B \to aS \mid bBB$
Trouver le langage généré par G puis l'AàP qui reconnaît L(G).

Réponse : $L(G) = \{\omega \in \{a, b\}^* / |\omega|_a = |\omega|_b\}$.
L'AàP qui reconnaît L(G) par pile vide est donné par $\Sigma = \{a, b\}$;
$\Gamma = \{S, A, B\}$;
$Q = \{q_0\}$;
$z_0 = S$;
Définition de δ :
$S \to aA$ donne $Sq_0 \to Aaq_0$;
$S \to bB$ donne $Sq_0 \to Bbq_0$;
$S \to \varepsilon$ donne $Sq_0 \to q_0$;
$A \to bS$ donne $Aq_0 \to Sbq_0$;
$A \to aAA$ donne $Aq_0 \to AAaq_0$;
$B \to aS$ donne $Bq_0 \to Saq_0$;
$B \to bBB$ donne $Bq_0 \to BBbaq_0$;
et $aq_0a \to q_0$ et $bq_0b \to q_0$.

3.10 Propriétés de fermeture de la classe des langages algébriques

La classe des langages algébriques possède est fermée par rapport aux opérations régulières, en d'autres termes, si L_1 et L_2 sont des langages algébriques alors :
$L_1.L_2$ est un langage algébrique ;

$L_1 \cup L_2$ est un langage algébrique ;
L_1^* est un langage algébrique ;
L_1^R est un langage algébrique.

Remarque : Contrairement à la classe des langages rationnels, la classe des langages algébriques n'est pas fermée (stable) par rapport l'intersection et au complément. Cependant, l'intersection d'un langage algébrique et d'un langage régulier est toujours algébrique.

Méthodes pour montrer qu'un langage est algébrique : Pour montrer qu'un langage est algébrique on peut choisir l'une des méthodes suivantes :

- Montrer que le langage est régulier, car l'ensemble des langages réguliers est inclus dans l'ensemble des langages algébriques ;
- Trouver un Automate à pile reconnaissant ce langage ;
- Trouver une grammaire de type 2 générant ce langage ;
- On peut exploiter les propriétés de fermeture des langages algébrique pour montrer qu'un langage est algébrique.

Chapitre 4

Langages semi-décidables et machines de Turing (MT)

Sommaire

Les machines abstraites reconnaissant les langages de type 1 (contextuels) et type 0 (semi-décidables) sont les machines de Turing. Les machines de Turing sont une extension des automates à une pile. Les principales différences sont :

1. Alors qu'un Aàp a le droit d'écrire et de lire uniquement le sommet de la pile, une MT peut déplacer, à sa guise, sa tête de lecture/écriture dans la mémoire, qu'on n'appelle donc plus une pile mais un ruban.
2. La seconde différence est que le mot à reconnaître est inscrit sur le ruban.

Ces deux extensions, qui paraissent minimes, suffisent à donner aux MT la puissance du décidable (calculable).
Les langages contextuels sont reconnus par un type particulier de machines de Turing appelées des machines de Turing linéairement bornées (ou Automates à Bornes Linéaires (ABL)) Un ABL est une machine de Turing qui vérifie les trois propriétés suivantes :

1. L'alphabet du ruban possède deux symboles particuliers qui servent comme marqueurs de fin à gauche et à droite ;
2. Sa tête de lecture/écriture ne peut écrire ou lire sur le ruban au-delà des marqueurs de fin ;
3. Les marqueurs de fin gauche et droite ne peuvent être déplacés au-delà de leurs positions respectivement à gauche et à droite.

Dans le présent chapitre nous commençons par introduire quelques notions relatives à la décidabilité, puis nous explicitons la relation entre les langages semi-décidables, les grammaires et les machines de Turing, enfin nous terminons par exposer la thèse de Church-Turing et la relation entre les machines de Turing et la complexité algorithmique.

4.1 Langages et problèmes décidables

La question à laquelle on s'intéresse est de construire (si cela est possible) un algorithme qui décide pour une instance donnée E, si une certaine propriété P est vraie ou non. Par exemple, décider pour un entier naturel s'il est premier ou non, ou-bien décider si un mot ω appartient à un langage L ou non.

Terminologie de problème ou langage décidable : Nous pouvons associer à n'importe quel problème (de décision) un langage et réciproquement.

En effet, à un problème est associé généralement implicitement une fonction de codage qui permet de coder les instances, c'est-à-dire les éléments de E, par un mot sur un certain alphabet Σ. On peut donc voir E comme un sous-ensemble de Σ^*, où Σ est un certain alphabet : au problème de décision P, on associe le langage L(P) correspondant à l'ensemble des mots codant une instance de E, qui appartient à E^+ :

4.2 Décidabilité des langages

4.2.1 Langage non décidable

Théorème : Il existe des problèmes de décision qui ne sont pas décidables.
La démonstration de ce théorème est faite dans le support de cours de M. PERIN A. cité dans le préambule. Il n'existe pas de grammaires syntagmatiques qui génèrent les langages non décidables.

Exemples de problèmes non décidables

1. **Le dixième problème de Hilbert :** David Hilbert a identifié 23 problèmes intéressants pour le 20ème siècle en 1900, le dixième problème est non décidable (prouvé en 1910). Dans ce problème, on s'intéresse à trouver si une équation polynomiale à coefficients entiers possède une solution entière, on appelle ce genre d'équations une équation diophantienne.
2. **La non décidabilité du problème de l'arrêt d'une MT :** Il a été démontré par Alan Turing en 1936 qu'il n'existe pas d'algorithme qui permette de décider si, étant donné une machine de Turing quelconque et un mot d'entrée, le calcul de celle-ci s'arrête ou non.

4.2.2 Langage semi-décidable

Définition : Un langage $L \subset \Sigma^*$ est dit semi-décidable (ou récursivement énumérable) si et seulement s'il existe un algorithme A, tel-que pour tout mot $u \in \Sigma^*$:

- Si $u \in L$, alors A termine sur u en produisant la sortie Vrai ;
- Si $u \notin L$, alors soit A termine sur u en produisant la sortie Faux, soit A ne termine pas.

4.2.3 Langage décidable

Définition 1 : Un langage $L \subset \Sigma^*$ est décidable (ou récursif, ou calculable) s'il existe un algorithme A qui, prenant un mot $u \in \Sigma^*$en entrée, termine et produit Vrai si $u \in L$ et Faux sinon. On dit alors que l'algorithme A décide le langage L.

Définition 2 : Un langage L est décidable si et seulement si L et son complément sont semi-décidables.

Remarque : Cette définition justifie la terminologie de semi-décidable, puisqu'un langage qui est semi-décidable et dont le complémentaire l'est aussi est décidable.

4.3 Propriétés de fermeture des classes des langages décidables et sem-décidables

Théorème : L'ensemble des langages semi-décidables est clos (fermé) par rapport à l'union et l'intersection.
Autrement dit, si L_1 et L_2 sont semi-décidables, alors $L_1 \cup L_2$ et $L_1 \cap L_2$ le sont aussi.

Théorème : L'ensemble des langages décidables est clos par rapport à l'union, l'intersection, et au complément.
Autrement dit, si L_1 et L_2 sont décidables, alors $L_1 \cup L_2$, $L_1 \cap L_2$ et le complément de L_1 le sont aussi.

4.4 Grammaires et langages semi-décidables

L'ensemble des langages semi-décidables inclut tous les langages qui peuvent être définis par une grammaire formelle, c'est aussi l'ensemble des langages acceptables par une machine de Turing. Nous pouvons, donc, conclure que l'ensemble des langages de type 0 est le même que l'ensemble des langages récursivement énumerables.
Il est équivalent de définir les langages récursivement énumérables comme les langages L pour lesquels il existe une machine de Turing qui reconnaît les mots de L, c'est-à-dire qui s'arrête dans un état d'acceptation pour tout mot de L. Cela ne préjuge en rien du comportement de la machine pour un mot qui n'est pas dans L.

4.5 Machines de Turing

Une MT est composée de d'une bande infinie ou semi-infinie, d'une partie de contrôle constituée d'un nombre fini d'états possibles, des transitions qui régissent les calculs de la machine et d'une tête de lecture/écriture. En fonction de la case courante et de l'état courant elle peut, écrire sur sa case courante, déplacer sa tête de lecture/écriture à gauche, à droite ou rester stationnaire et éventuellement changer d'état. La figure 4.1 illustre la structure générale d'une machine de Turing.

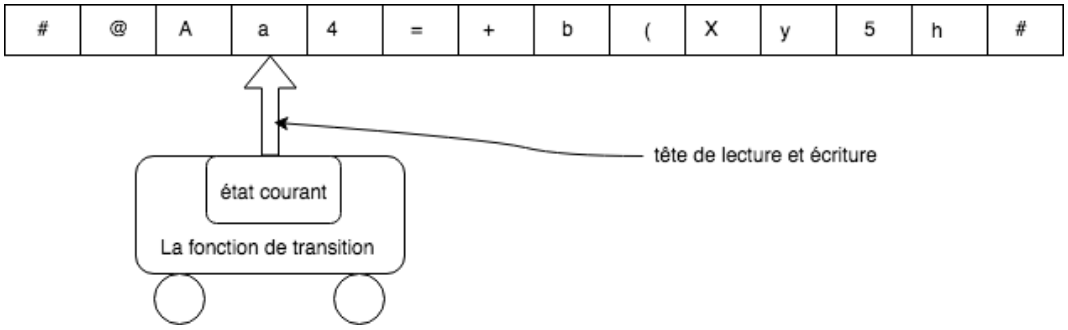

FIGURE 4.1 – Structure d'une machine de Turing.

4.5.1 Définition formelle

Il existe de nombreuses variantes, toutes équivalentes, des machines de Turing. Dans ce cours, on adopte les conventions suivantes :

Une MT est définie formellement par un septuplé $(\Sigma, \Gamma, Q, q_0, \delta, \#, F)$ où :

- Σ est l'alphabet d'entrée ;
- Γ est l'ensemble des symboles auxiliaires. $\Gamma \cup \Sigma$ est l'alphabet du ruban ;
- Q est un ensemble fini d'états ;
- q_0 est l'état initial ;
- $\#$ est le symbole utilisé pour les cases vides du ruban ;
- F est l'ensemble des états finaux (d'acceptation) ;
- δ est la fonction de transition, elle est définie de $Q \times (\Gamma \cup \Sigma)$ dans $(\Gamma \cup \Sigma) \times \{G, D, S\} \times Q$. G, D et S signifient respectivement le déplacement de la tête de lecture/écriture à gauche, à droite d'une case ou rester stationnaire (ne pas bouger).

On suppose que, initialement, la tête de lecture/écriture est placée sur le premier caractère dans le ruban.

Exemple : Soit la la machine de Turing $(\Sigma, \Gamma, Q, q_0, \delta, \#, F)$ définie par :
$\Sigma = \{a, b\}$, $\Gamma = \{\#, 0\}$, $Q = \{q_0, q_1, q_f\}$, l'état initial est q_0, $F = \{q_f\}$ et la fonction de transition δ est définie comme suit :

1. $\delta(q_0, a) = (0, D, q_1)$;
2. $\delta(q_0, b) = (b, D, q_0)$;
3. $\delta(q_1, a) = (0, D, q_0)$;
4. $\delta(q_1, b) = (b, D, q_1)$;
5. $\delta(q_1, \#) = (\#, S, q_f)$;

Nous cherchons à savoir si les mots abaa, baba sont acceptés par cette machine de Turing.

Réponse : Nous étudions si les mots sont acceptés par les tableaux qui suivent :

Ruban	état	inst	interprétation
..#, (b), a, b, a, #..	q_0	2	
..#, b, (a), b, a, #..	q_0	1	
..#, b, a, (b), a, #..	q_1	4	
..#, b, a, b, (a), #..	q_1	3	
..#, b, a, b, a, (#)..	q_0	/	Aucune instruction exécutable ($\notin$)

Ruban	état	inst	interprétation
..#, (a), b, a, a, #..	q_0	1	
..#, a, (b), a, a, #..	q_1	4	
..#, a, b, (a), a, #..	q_1	3	
..#, a, b, a, (a), #..	q_0	1	
..#, a, b, a, a, (#)..	q_1	5	Passer à l'état q_f, le mot est accepté

4.5.2 Définition d'une configuration

Une configuration d'une machine de Turing est l'état global de la machine à un instant donné. Elle contient : l'état courant, le contenu du ruban et la position de la tête de lecture/écriture. Si la machine se trouve à l'état q, alors la configuration est uqv, avec u ce qui se trouve strictement à gauche de la tête de lecture/écriture et v ce qui s'y trouve à droite (y compris ce qu'il y a sous la tête de lecture/écriture).
La configuration initiale d'une machine de Turing est donnée par $q_0\omega$, où q_0 est l'état initial et ω est la donnée inscrite initialement sur le ruban.

4.5.3 Dérivation directe

On dit qu'une configuration C_1 dérive directement une configuration C_2, et on note $C_1 \vdash C_2$ (aussi $C_1 \Rightarrow C_2$) si et seulement s'il existe une transition de la fonction δ qui fait passer la MT de la configuration C_1 à la configuration C_2.

4.5.4 Dérivation indirecte

On dit qu'une configuration C_1 dérive indirectement une configuration C_2, et on note $C_1 \vdash^* C_2$ (aussi $C_1 \Rightarrow^* C_2$) si et seulement s'il existe 0, 1 ou plusieurs dérivations directes qui font passer la MT de la configuration C_1 à la configuration C_2. $\vdash^*$ est la fermeture réflexive et transitive de la relation $\vdash$.

4.5.5 Configuration bloquante et configuration acceptante

Une configuration C_1 est dite bloquante s'il n'existe pas de configuration C_2 tel-que $C_1 \vdash C_2$.

Une configuration $C = uqv$ est dite acceptante si q est un état final.

4.5.6 Langage accepté par une machine de Turing

On dit qu'une machine de Turing MT reconnaît le langage L si et seulement si $\forall \omega \in L, q_0\omega \vdash^* C$ où q_0 est l'état initial et C est une configuration acceptante.

Remarque : Un langage peut être reconnu par plusieurs machines de Turing.

4.6 Modes d'utilisation d'une machine de Turing

4.6.1 Mode accepteur

Dans le mode accepteur, on fournit un mot d'entrée à la machine et celle-ci répond par oui si le mot appartient à un langage particulier (elle passe à un état final), ou par non s'il n'appartient pas (passe à un état bloquant). Nous attirons l'attention de l'étudiant que dans les langages semi-décidables (type 0), il est possible que la machine ne s'arrête jamais et ne réponde ni par oui ni par non.

4.6.2 Mode calculateur

Dans le mode calculateur, on fournit un mot d'entrée à la machine et celle-ci retourne un ou plusieurs mots de sortie, dans ce mode d'utilisation, la tête de lecture/écriture doit être remise au début du mot résultat.

Exemple de machine de Turing en mode accepteur : Soit la la machine de Turing MT_1 $(\Sigma, \Gamma, Q, q_0, \delta, \#, F)$ suivante :
$\Sigma = \{a, b, c\}$, $\Gamma = \{\#, 0\}$, $Q = \{q_0, q_1, q_2, q_3, q_4, q_6, q_f\}$, l'état initial est q_0, $F = \{q_f\}$ et la fonction de transition est définie comme suit :

1. $\delta(q_0, a) = (0, D, q_1)$;
2. $\delta(q_1, a) = (a, D, q_1)$;
3. $\delta(q_1, b) = (0, D, q_2)$;
4. $\delta(q_2, b) = (b, D, q_2)$;
5. $\delta(q_2, c) = (0, G, q_3)$;
6. $\delta(q_3, b) = (b, G, q_4)$;
7. $\delta(q_3, 0) = (0, G, q_5)$;
8. $\delta(q_4, a) = (a, G, q_4)$;
9. $\delta(q_4, 0) = (0, D, q_0)$;
10. $\delta(q_5, 0) = (0, G, q_5)$;
11. $\delta(q_5, \#) = (\#, D, q_6)$;
12. $\delta(q_6, 0) = (0, D, q_6)$;
13. $\delta(q_6, \#) = (\#, S, q_f)$.

Le langage accepté par MT_1 est $L(MT_1) = \{a^n b^n c^n / n \geq 1\}$.

Exemple de machine de Turing en mode calculateur : Soit la la machine de Turing MT_2 $(\Sigma, \Gamma, Q, q_0, \delta, \#, F)$ suivante :
$\Sigma = \{0, 1\}$, $\Gamma = \{\#\}$, $Q = \{q_0, q_1, q_2, q_f\}$, l'état initial est q_0, $F = \{q_f\}$ et la fonction de transition δ est définie comme suit :

1. $\delta(q_0, 1) = (1, D, q_0)$;
2. $\delta(q_0, 0) = (0, D, q_0)$;
3. $\delta(q_0, \#) = (\#, G, q_1)$;

4. $\delta(q_1, 0) = (1, S, q_2)$;
5. $\delta(q_1, 1) = (0, G, q_1)$;
6. $\delta(q_1, \#) = (1, S, q_f)$;
7. $\delta(q_2, 0) = (0, G, q_2)$;
8. $\delta(q_2, 1) = (1, G, q_2)$;
9. $\delta(q_2, \#) = (\#, D, q_f)$.

La machine de Turing MT_2 calcule la fonction successeur d'un nombre binaire. $f : x \mapsto x + 1$

Notation : Pour des raisons de légèreté d'écriture et d'efficacité, à la place de $\delta(q, x) = (y, m, p)$ nous écrivons $qx \to ymp$.

4.7 ABL et langages contextuels

Les automates linéairement bornés reconnaissent exactement la classe des langages contextuels. Ce type d'automates est un cas particulier des machines de Turing, nous attirons l'attention de l'étudiant que dans une grammaire monotone (grammaire contextuelle), une étape d'une dérivation allonge toujours le mot produit. Si l'on essaie donc de réduire un mot en l'axiome, chaque étape revient à raccourcir le mot. C'est pourquoi une mémoire bornée suffit. Nous déduisons alors que ce type de machine de Turing s'arrête toujours, par conséquent les langages de type 1 sont décidables. La taille exploitée du ruban par l'ABL sera une fonction linéaire de la taille du mot inscrit initialement.

4.8 Thèse de Church-Turing

De 1931 à 1936, Alan Turing présente ses machines, Jacques Herbrand et Kurt Gödel les systèmes d'équations HG, Stephen Cole Kleene les fonctions μ-récursives et Alonso Church propose le λ-calcul. Ces propositions sont des tentatives très différentes pour définir des fonctions calculables, elles se sont avérées équivalentes. La preuve de ces équivalences consiste à trouver un codage d'un système dans un autre et réciproquement.
Puisque des modèles de calcul très différents conduisent à une notion équivalente de fonctions calculables, il est raisonnable de penser, mais ce n'est qu'une hypothèse, que

la notion de fonctions calculables est indépendante du modèle de calcul. Puisqu'on ne parvient pas à concevoir un modèle de calcul plus puissant que ceux déjà connus, on adopte ces modèles comme définitions de la notion de fonctions calculables.
En théorie de la calculabilité, on emploie aussi le terme historique de fonctions récursives. On lui préférera le terme fonctions calculables afin de ne pas confondre avec la notion de fonctions récursives en programmation qui sont des fonctions qui s'appellent elle-mêmes.

Énoncé de la thèse de Church-Turing : La classe des fonctions calculables par machines de Turing est égale à la classe des fonctions calculables par un algorithme quelconque.

4.9 Déterminisme des machines de Turing et complexité algorithmique

4.9.1 Machines de Turing déterministes et classe des problèmes P

Ce qui distingue une MTD d'une MTND est la nature de la fonction de transition δ. Pour une MTD donnée, la fonction de transition δ associe à chaque couple $(x, q) \in (\Gamma \cup \Sigma) \times Q$ au plus un triplé $(y, p, q) \in (\Gamma \cup \Sigma) \times \{G, D, S\} \times Q$. Autrement dit, à chaque couple (x, q), où q désigne l'état courant et x le symbole de la case courante, la fonction de transition associe au plus un triplé (y, p, q), où p désigne l'état de la MTD après le pas, y le symbole que la tête de lecture/ écriture a substitué à x dans la case courante, et m le mouvement effectué par cette tête après avoir écrit y . On notera que δ n'est pas nécessairement une application : certains couples peuvent ne pas avoir d'image.

Proposition : La classe P (Polynomiale) est l'ensemble des problèmes de décision que l'on peut résoudre à l'aide d'une MTD en un temps polynomial.

4.9.2 Machines de Turing non déterministes et classe des problèmes NP

Comme nous l'avons déjà souligné auparavant, la seule différence entre une MTD et une MTND porte sur la fonction de transition. La fonction d'une MTND peut associer à un couple $(x, q) \in (\Gamma \cup \Sigma) \times Q$ plusieurs triplés $(y, p, q) \in (\Gamma \cup \Sigma) \times \{G, D, S\} \times Q$. C'est là que réside le caractère aléatoire de la MTND : à chaque pas du fonctionnement de la MTND, on choisit au hasard le triplé définissant l'instruction que l'on va exécuter parmi les possibilités associées.

Proposition : La classe NP (Non polynomiale) est l'ensemble des problèmes de décision dont on peut résoudre les données admettant la réponse "oui" en un temps polynomial à l'aide d'une MTND.