Théorie des langages

Christine Solnon

Table des matières

1 Motivations **2**

2 Alphabets, Langages et Grammaires **3**

2.1 Alphabets et mots 3
2.2 Langages 5
2.3 Grammaires 8
2.4 Types de grammaires 12

3 Langages réguliers et Automates finis **15**

3.1 Grammaires régulières et langages réguliers 15
3.2 Automates Finis Indéterministes 16
3.3 Automates Finis Déterministes 18
3.4 Equivalence entre AFI et AFD 19
3.5 Equivalence entre automates finis et langages réguliers 21
3.6 Expressions régulières 22
3.7 Quelques propriétés des langages réguliers 23

4 Langages hors-contexte et Automates à pile **24**

4.1 Arbres syntaxiques 24
4.2 La forme de BACKUS-NAUR d'une grammaire 26
4.3 Propriétés de fermeture des langages hors-contexte 26
4.4 Automates à pile 27
4.5 Automates à pile déterministes 29
4.6 Automates à pile et langages hors-contexte 29

1 Motivations

L'objet de ce cours est une initiation à la théorie des langages formels. De manière générale, les langages sont les supports naturels de communication. Ils permettent aux hommes d'échanger des informations et des idées, ils leur permettent également de communiquer avec les machines. Les langages utilisés dans la vie de tous les jours entre êtres humains sont dits naturels. Ils sont généralement informels et ambigus et demandent toute la subtilité d'un cerveau humain pour être interprétés correctement. Les langages créés par l'homme pour communiquer avec les ordinateurs sont des langages artificiels. Ils doivent être formalisés et non ambigus pour pouvoir être interprétés par une machine.

Au départ, un ordinateur ne comprend qu'un seul langage, pour lequel il a été conçu : son langage machine. Pour communiquer avec des langages plus évolués, il est nécessaire d'utiliser un interprête (qui traduit inter-activement les instructions entrées au clavier), ou bien un compilateur (qui traduit tout un programme). L'interprétation ou la compilation d'un texte se décomposent généralement en trois étapes.

1. Une première phase d'*analyse lexicale* permet de décomposer le texte en entités élémentaires appelées lexèmes (token en anglais).
2. Une deuxième phase d'*analyse syntaxique* permet de reconnaître des combinaisons de lexèmes formant des entités syntaxiques.
3. Une troisième phase d'*analyse sémantique* permet de générer le code objet directement compréhensible par la machine (ou bien un code intermédiaire qui devra être de nouveau traduit dans un code machine).

Considérons par exemple, le (morceau de) texte C suivant : `cpt = i + 3.14;`

1. L'analyse lexicale permet d'identifier les lexèmes suivants : un `IDENTIFICATEUR` de valeur `cpt`, un `OPERATEUR` de valeur `=`, un `IDENTIFICATEUR` de valeur `i`, un `OPERATEUR` de valeur `+`, un `REEL` de valeur 3.14 et un `POINT VIRGULE`.
2. L'analyse syntaxique permet de reconnaître que cette combinaison de lexèmes forme une instruction C syntaxiquement correcte, et qu'il s'agit d'une affectation entre la variable d'identificateur `cpt` et l'expression arithmétique résultant de l'addition de la variable d'identificateur `i` avec le réel 3.14.
3. Enfin, l'analyse sémantique vérifie le bon typage des variables `cpt` et `i`, puis génère le code objet correspondant à cette instruction.

Les phases d'analyse lexicale et syntaxique constituent en fait un même problème (à deux niveaux différents). Dans les deux cas, il s'agit de reconnaître une combinaison valide d'entités : une combinaison de caractères formant des lexèmes pour l'analyse lexicale, et une combinaison de lexèmes formant des programmes pour l'analyse syntaxique. La théorie des langages permet de résoudre ce type de problème.

Plan du cours

En théorie des langages, l'ensemble des entités élémentaires est appelé l'alphabet. Une combinaison d'entités élémentaires est appelé un mot. Un ensemble de mots est appelé un langage et est décrit par une grammaire. A partir d'une grammaire, on peut construire une procédure effective (appelée automate) permettant de décider si un mot fait partie du langage. Dans la partie 2 de ce cours, nous définissons ces différentes notions, et nous décrivons certaines de leurs propriétés.

Il existe différentes classes de langages, correspondant à différentes classes de grammaires et d'automates. Dans la partie 3, nous étudions la classe des langages réguliers, correspondant aux grammaires régulières et aux automates finis. Cette classe de grammaire est typiquement utilisée pour décrire les entités lexicales d'un langage de programmation.

Dans la partie 4, nous étudions la classe des langages hors contexte, correspondant aux grammaires hors contexte et aux automates à pile. Cette classe de grammaire, plus puissante que la classe des grammaires régulières, est typiquement utilisée pour décrire la syntaxe d'un langage de programmation.

2 Alphabets, Langages et Grammaires

2.1 Alphabets et mots

En théorie des langages, l'ensemble des entités élémentaires est appelé l'alphabet. Une combinaison d'entités élémentaires est appelé un mot.

Définition (Alphabet) : Un alphabet, noté $\mathcal{A}$, est un ensemble fini non vide de symboles

Exemples d'alphabets :

$\mathcal{A}_1 = \{ \bullet, \star, \diamond \}$

$\mathcal{A}_2 = \{$ a, b, c, ..., z $\}$

$\mathcal{A}_3 = \{$ if, then, else, id, nb, $=$, $+$ $\}$

Définition (Mot) : Un mot, défini sur un alphabet $\mathcal{A}$, est une suite finie d'éléments de $\mathcal{A}$.

Exemples de mots :

- *sur l'alphabet* $\mathcal{A}_1$*, le mot* $\bullet \bullet \star$
- *sur l'alphabet* $\mathcal{A}_2$*, le mot* if
- *sur l'alphabet* $\mathcal{A}_3$*, le mot* if id $=$ nb

Terminologie :

— Lors de l'analyse lexicale d'un programme, l'alphabet est l'ensemble des symboles du clavier, tandis que les mots sont les mots clés, les identificateurs, les nombres, les opérateurs, ... et sont généralement appelés lexèmes.

— Lors de l'analyse syntaxique d'un programme, les éléments de base de l'alphabet sont les mots clés, les identificateurs, les nombres, les opérateurs, ... (autrement dit, les lexèmes de l'analyse lexicale), tandis qu'un mot est une suite de lexèmes et forme un programme.

— D'une facon plus générale, lorsque les éléments de l'ensemble de base $\mathcal{A}$ sont des mots au sens linguistique, on emploie le terme de vocabulaire à la place d'alphabet pour désigner $\mathcal{A}$, et le terme de phrase (ou chaîne) à la place de mot pour désigner une séquence finie de mots linguistiques.

Définition (Longueur d'un mot) : La longueur d'un mot u défini sur un alphabet $\mathcal{A}$, notée $|u|$, est le nombre de symboles qui composent u.

Par exemple :
- sur l'alphabet $\mathcal{A}_1$, $\mid \bullet \bullet \star \mid = 3$
- sur l'alphabet $\mathcal{A}_2$, $\mid if \mid = 2$
- sur l'alphabet $\mathcal{A}_3$, $\mid if \;\; id = nb \mid = 4$

Définition (Mot vide) : le mot vide, noté ϵ, est défini sur tous les alphabets et est le mot de longueur 0 (autrement dit, $|\epsilon| = 0$).

Définition ($\mathcal{A}^+$) : on note $\mathcal{A}^+$ l'ensemble des mots de longueur supérieure ou égale à 1 que l'on peut construire à partir de l'alphabet $\mathcal{A}$.

Définition ($\mathcal{A}^*$) : on note $\mathcal{A}^*$ l'ensemble des mots que l'on peut construire à partir de $\mathcal{A}$, y compris le mot vide : $\mathcal{A}^* = \{\epsilon\} \cup \mathcal{A}^+$

Définition (Concaténation) : Soient deux mots u et v définis sur un alphabet $\mathcal{A}$. La concaténation de u avec v, notée $u.v$ ou simplement uv s'il n'y a pas d'ambigüité, est le mot formé en faisant suivre les symboles de u par les symboles de v. On notera u^n le mot u concaténé n fois ($u^0 = \epsilon$, $u^n = u.(u^{n-1})$ pour $n \geq 1$).

Par exemple, sur l'alphabet $\mathcal{A}_2$*, si* $u = aabb$ *et* $v = cc$*, alors* $u.v = aabbcc$ *et* $u^3 = aabbaabbaabb$.

Propriétés : La concaténation est associative mais non commutative. La concaténation est une loi de composition interne de $\mathcal{A}^*$ et ϵ est son élément neutre. Par conséquent, $(\mathcal{A}^*, .)$ est un monoïde.

Exercice : Soit l'alphabet $\mathcal{A} = \{a, b\}$.

1. Etant donnés les mots $u = aa$ et $v = bab$, écrire les mots uv, $(uv)^2$ et u^3v.
2. Enoncer tous les mots de longueur 2 définis sur $\mathcal{A}$.
3. Soient les ensembles

$$\begin{aligned} E_1 = &\quad \{u.v / u \in \mathcal{A}^+, v \in \mathcal{A}^+\} \\ E_2 = &\quad \{u.v / u \in \mathcal{A}^+, v \in \mathcal{A}^*\} \\ E_3 = &\quad \{u.v / u \in \mathcal{A}^*, v \in \mathcal{A}^*\} \end{aligned}$$

   A quoi correspondent ces ensembles ?

Correction :

1. $uv = aabab$, $(uv)^2 = aababaabab$ et $u^3v = aaaaaabab$.
2. Mots de longueur $2 = \{aa, ab, ba, bb\}$
3. $E_1 = \{u \in \mathcal{A}^* / |u| \geq 2\}$ = ensemble des mots d'au moins 2 symboles
   $E_2 = \mathcal{A}^+$
   $E_3 = \mathcal{A}^*$

Définition (Préfixe, suffixe et facteur) : Soient deux mots u et v définis sur un alphabet $\mathcal{A}$.
- u est un préfixe de v si et seulement si $\exists w \in \mathcal{A}^*$ tel que $uw = v$;
- u est un suffixe de v si et seulement si $\exists w \in \mathcal{A}^*$ tel que $wu = v$;
- u est un facteur de v si et seulement si $\exists w_1 \in \mathcal{A}^*$, $\exists w_2 \in \mathcal{A}^*$ tels que $w_1uw_2 = v$.

Exercice : Montrer que les relations “être-préfixe-de”, “être-suffixe-de” et “être-facteur-de” sont des relations d’ordre partiel sur $\mathcal{A}^*$, c’est-à-dire qu’elles sont transitives, antisymétriques et réflexives.

Correction pour “être-préfixe-de” :

- Transitivité : soient trois mots u, v et w définis sur $\mathcal{A}$ tels que u est un préfixe de v et v est un préfixe de w. Montrons que u est un préfixe de w :
  - u est un préfixe de $v \Rightarrow \exists u' \in \mathcal{A}^*$ tel que $uu' = v$
  - v est un préfixe de $w \Rightarrow \exists v' \in \mathcal{A}^*$ tel que $vv' = w$

  Par conséquent, $w = uu'v'$ et donc u est un préfixe de w.
- Antisymétrie : soient deux mots u et v définis sur $\mathcal{A}$ tels que u est un préfixe de v et v est un préfixe de u. Montrons que u est égal à v :
  - u est un préfixe de $v \Rightarrow \exists u' \in \mathcal{A}^*$ tel que $uu' = v$
  - v est un préfixe de $u \Rightarrow \exists v' \in \mathcal{A}^*$ tel que $vv' = u$

  Par conséquent, $uu'v' = u$ et donc $u' = \epsilon$ et $v' = \epsilon$ et $u = v$.
- Réflexivité : pour tout mot u défini sur $\mathcal{A}$, on a u est un préfixe de u car $u = u.\epsilon$ et $\epsilon \in \mathcal{A}^*$.

Exercice : On considère les ensembles de mots E_1 et E_2 définis sur l’alphabet $\mathcal{A} = \{0, 1, 2\}$ de la façon suivante :
- E_1 est l’ensemble des mots de longueur paire,
- E_2 est l’ensemble des mots comportant autant de 0 que de 1 et autant de 1 que de 2.
Définir de façon plus formelle ces deux ensembles et déterminer pour chacun d’eux si la concaténation est une loi interne et si le mot vide en est un élément.

Correction :

- $E_1 = \{u \in \mathcal{A}^* / \exists l \in \mathbb{N}, |u| = 2l\}$
  - La concaténation est une loi interne pour E_1 car pour tout couple de mots $(u, v) \in E_1^2$, $|uv| = |u| + |v| = 2l + 2l' = 2(l + l')$.
  - $\epsilon \in E_1$ car $|\epsilon| = 2 * 0$
- Pour définir formellement l’ensemble E_2, il est nécessaire d’introduire la notion de permutations d’un mot. L’ensemble des permutations d’un mot u est l’ensemble de tous les mots que l’on peut former en ré-arrangeant les symboles qui composent u de toutes les façons possibles. Plus formellement, on peut définir cet ensemble récursivement de la façon suivante :
  - permutations(ϵ) $= \{\epsilon\}$
  - pour tout mot $u \in \mathcal{A}^+$ commençant par un symbole $a \in \mathcal{A}$ et se terminant par une suite de symboles $u' \in \mathcal{A}^*$ (tel que $u = a.u'$), permutations(u) $= \{v'.a.v'' / v'v'' \in$ permutations(u')$\}$

  On peut alors définir E_2 de la façon suivante : $E_2 = \{u / \exists n \in \mathbb{N}, u \in \text{permutations}(0^n 1^n 2^n)\}$.
  La concaténation est une loi interne pour E_2 et $\epsilon \in E_2$.

2.2 Langages

Définition (Langage) : Un langage, défini sur un alphabet $\mathcal{A}$, est un ensemble de mots définis sur $\mathcal{A}$. Autrement dit, un langage est un sous-ensemble de $\mathcal{A}^*$.

Deux langages particuliers sont indépendants de l’alphabet $\mathcal{A}$:
- le langage vide ($\mathcal{L} = \emptyset$),

- le langage contenant le seul mot vide ($\mathcal{L} = \{\epsilon\}$).

Opérations ensemblistes définies sur les langages : Soient deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$ respectivement définis sur les alphabets $\mathcal{A}_1$ et $\mathcal{A}_2$:

— L'union de $\mathcal{L}_1$ et $\mathcal{L}_2$ est le langage défini sur $\mathcal{A}_1 \cup \mathcal{A}_2$ contenant tous les mots qui sont soit contenus dans $\mathcal{L}_1$, soit contenus dans $\mathcal{L}_2$:

$$\mathcal{L}_1 \cup \mathcal{L}_2 = \{u/u \in \mathcal{L}_1 \text{ ou } u \in \mathcal{L}_2\}$$

— L'intersection de $\mathcal{L}_1$ et $\mathcal{L}_2$ est le langage défini sur $\mathcal{A}_1 \cap \mathcal{A}_2$ contenant tous les mots qui sont contenus à la fois dans $\mathcal{L}_1$ et dans $\mathcal{L}_2$:

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \{u/u \in \mathcal{L}_1 \text{ et } u \in \mathcal{L}_2\}$$

— Le complément de $\mathcal{L}_1$ est le langage défini sur $\mathcal{A}_1$ contenant tous les mots qui ne sont pas dans $\mathcal{L}_1$:

$$C(\mathcal{L}_1) = \{u/u \in \mathcal{A}_1^* \text{ et } u \notin \mathcal{L}_1\}$$

— La différence de $\mathcal{L}_1$ et $\mathcal{L}_2$ est le langage défini sur $\mathcal{A}_1$ contenant tous les mots de $\mathcal{L}_1$ qui ne sont pas dans $\mathcal{L}_2$:

$$\mathcal{L}_1 - \mathcal{L}_2 = \{u/u \in \mathcal{L}_1 \text{ et } u \notin \mathcal{L}_2\}$$

Définition (Produit de deux langages) : Le produit ou concaténation de deux langages $\mathcal{L}_1$ et $\mathcal{L}_2$, respectivement définis sur les alphabets $\mathcal{A}_1$ et $\mathcal{A}_2$, est le langage défini sur $\mathcal{A}_1 \cup \mathcal{A}_2$ contenant tous les mots formés d'un mot de $\mathcal{L}_1$ suivi d'un mot de $\mathcal{L}_2$:

$$\mathcal{L}_1.\mathcal{L}_2 = \{uv/u \in \mathcal{L}_1 \text{ et } v \in \mathcal{L}_2\}$$

Le produit de langages est associatif, mais non commutatif.

Considérons par exemple les deux langages $\mathcal{L}_1 = \{00, 11\}$ *et* $\mathcal{L}_2 = \{0, 1, 01\}$ *définis sur* $\{0, 1\}$.
$\mathcal{L}_1.\mathcal{L}_2 = \{000, 001, 0001, 110, 111, 1101\}$

Définition (Puissances d'un langage) : Les puissances successives d'un langage $\mathcal{L}$ sont définies récursivement par
- $\mathcal{L}^0 = \{\epsilon\}$,
- $\mathcal{L}^n = \mathcal{L}.\mathcal{L}^{n-1}$ pour $n \geq 1$.

Par exemple, si $\mathcal{L}_1 = \{00, 11\}$*, alors* $\mathcal{L}_1^2 = \{0000, 0011, 1100, 1111\}$

Définition (Fermeture itérative d'un langage) : La fermeture itérative d'un langage $\mathcal{L}$ (ou fermeture de Kleene ou itéré de $\mathcal{L}$) est l'ensemble des mots formés par une concaténation de mots de $\mathcal{L}$:

$$\mathcal{L}^* = \{u/\exists k \geq 0 \text{ et } u_1, ..., u_k \in \mathcal{L} \text{ tels que } u = u_1 u_2 ... u_k\}$$

Autrement dit, $\mathcal{L}^* = \cup_{i=0}^{\infty} \mathcal{L}^i$
De même, on définit $\mathcal{L}^+ = \cup_{i=1}^{\infty} \mathcal{L}^i$

Description d'un langage :

— Un langage fini peut être décrit par l'énumération des mots qui le composent.
— Certains langages infinis peuvent être décrits par l'application d'opérations à des langages plus simples.
— Certains langages infinis peuvent être décrits par un ensemble de règles appelé grammaire (voir la section suivante).
— Enfin, certains langages infinis ne peuvent pas être décrits, ni par l'application d'opérations, ni par un ensemble de règles. On parle alors de langage indécidable. On peut noter que si un langage est indécidable, alors il n'existe pas d'algorithme permettant de déterminer si un mot donné appartient à ce langage. On dit alors que le problème est indécidable. Par exemple, le langage des programmes C++ qui "terminent" (qui ne bouclent pas indéfiniment) ne peut être décrit par des règles formelles : ce langage est indécidable et le problème consistant à déterminer si un programme C++ donné termine est un problème indécidable, pour lequel il n'existe pas d'algorithme (ce problème est plus connu sous le nom de "problème de l'arrêt de la machine de Turing").

Exercice : Sur l'alphabet $\mathcal{A} = \{0, 1\}$, on considère les langages $\mathcal{L}_1$ et $\mathcal{L}_2$ définis par

$$\begin{aligned} \mathcal{L}_1 = &\quad \{01^n / n \in \mathbb{N}\} \\ \mathcal{L}_2 = &\quad \{0^n 1 / n \in \mathbb{N}\} \end{aligned}$$

Définir les langages $\mathcal{L}_1\mathcal{L}_2$, $\mathcal{L}_1 \cap \mathcal{L}_2$ et $\mathcal{L}_1^2$.

Correction :

— $\mathcal{L}_1\mathcal{L}_2 = \{01^n 0^m 1 / n \in \mathbb{N}, m \in \mathbb{N}\}$
— $\mathcal{L}_1 \cap \mathcal{L}_2 = \{01\}$
— $\mathcal{L}_1^2 = \{01^n 01^m / n \in \mathbb{N}, m \in \mathbb{N}\}$

Exercice : Sur l'alphabet $\mathcal{A} = \{a, b\}$, on considère le langage $\mathcal{L}_1$ des mots formés de n fois la lettre a suivi de n fois la lettre b, et le langage $\mathcal{L}_2$ des mots comportant autant de a que de b.
- Définir formellement ces deux langages.
- Que sont les langages suivants : $\mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1^2$, $\mathcal{L}_2^2$?
- Que peut-on dire de $\mathcal{L}_1^*$ et $\mathcal{L}_2^*$ par rapport à $\mathcal{L}_1$ et $\mathcal{L}_2$?

Correction :

— $\mathcal{L}_1 = \{a^n b^n / n \in \mathbb{N}\}$
— $\mathcal{L}_2 = \{u / \exists n \in \mathbb{N}, u \in \text{permutations}(a^n b^n)\}$
— $\mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}_2$ et $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathcal{L}_1$ car $\mathcal{L}_1 \subset \mathcal{L}_2$
— $\mathcal{L}_1^2 = \{a^n b^n a^m b^m / n \in \mathbb{N}, m \in \mathbb{N}\}$
— $\mathcal{L}_2^2 = \mathcal{L}_2$
— $\mathcal{L}_1 \subset \mathcal{L}_1^* \subset \mathcal{L}_2$
— $\mathcal{L}_2^* = \mathcal{L}_2$

2.3 Grammaires

Un langage peut être décrit par un certain nombre de règles. Cette vue du concept de langage a son origine dans des essais de formalisation du langage naturel. Le but était de donner une description précise des règles permettant de construire les phrases correctes d'une langue.

Prenons par exemple le sous-ensemble suivant de la grammaire francaise :

— le vocabulaire est défini par l'ensemble :

$$T = \{ \textit{le, la, fille, jouet, regarde} \}$$

— les catégories syntaxiques sont :

la phrase, notée PH
le groupe nominal, noté GN
le verbe, noté V
le déterminant, noté D
le nom, noté N

— les règles permettant de combiner des éléments du vocabulaire et des catégories syntaxiques pour construire des catégories syntaxiques sont les suivantes :

$PH \to GN\ V\ GN$	$N \to fille$
$GN \to D\ N$	$N \to jouet$
$D \to le$	$V \to regarde$
$D \to la$	

où le symbole $\to$ est une abréviation de "peut être composé de".

— la catégorie syntaxique de départ est la phrase *PH*.

La phrase "la fille regarde le jouet" est une phrase correcte pour la grammaire envisagée, comme le montre l'analyse suivante :

$$PH \Rightarrow GN\ V\ GN \Rightarrow D\ N\ V\ GN \Rightarrow la\ N\ V\ GN \Rightarrow la\ fille\ V\ GN \Rightarrow la\ fille\ regarde\ GN$$
$$\Rightarrow la\ fille\ regarde\ D\ N \Rightarrow la\ fille\ regarde\ le\ N \Rightarrow la\ fille\ regarde\ le\ jouet$$

où le symbole $\Rightarrow$ est une abréviation de "se dérive en".

Notons que :

1. La grammaire considérée ne prend pas en compte certains aspects du francais, comme les accords de genre.
2. "le jouet regarde la fille" est aussi une phrase syntaxiquement correcte, mais dont la sémantique n'est pas assurée.

La fonction d'une grammaire telle que celle que nous venons de donner est double : la grammaire indique comment construire des phrases appartenant au langage (fonctionnement en production) ; la grammaire permet également de décider si une phrase donnée appartient ou non au langage (fonctionnement en reconnaissance).

Dans le cas d'un langage de programmation, on se sert d'une grammaire pour décrire les entités du langage. La forme de Backus-Naur (BNF), souvent utilisée pour décrire la syntaxe des langages de programmation, est en fait une grammaire au sens où nous allons le définir.

Définition (Grammaire) : Une grammaire est un quadruplet $G = (T, N, S, R)$ tel que

— T est le vocabulaire terminal, c'est-à-dire l'alphabet sur lequel est défini le langage.

— N est le vocabulaire non terminal, c'est-à-dire l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération. Un symbole non terminal désigne une "catégorie syntaxique".

— R est un ensemble de règles dites de réécriture ou de production de la forme :

$$u1 \to u2, \text{ avec } u1 \in (N \cup T)^+ \text{ et } u2 \in (N \cup T)^*$$

La signification intuitive de ces règles est que la suite non vide de symboles terminaux ou non terminaux $u1$ peut être remplacée par la suite éventuellement vide de symboles terminaux ou non terminaux $u2$.

— $S \in N$ est le symbole de départ ou axiome. C'est à partir de ce symbole non terminal que l'on commencera la génération de mots au moyen des règles de la grammaire.

Terminologie :

— une suite de symboles terminaux et non terminaux (un élément de $(N \cup T)^*$) est appelée une forme.

— une règle $u1 \to u$ telle que $u \in T^*$ est appelée une règle terminale.

Notation : Lorsque plusieurs règles de grammaire ont une même forme en partie gauche, on pourra "factoriser" ces différentes règles en séparant les parties droites par des traits verticaux. Par exemple, l'ensemble de règles $S \to ab,\ S \to aSb,\ S \to c$ pourra s'écrire $S \to ab \mid aSb \mid c$.

Le langage défini, ou généré, par une grammaire est l'ensemble des mots qui peuvent être obtenus à partir du symbole de départ par application des règles de la grammaire. Plus formellement, on introduit les notions de dérivation entre formes, d'abord en une étape, ensuite en plusieurs étapes. Enfin, on définit le langage généré par une grammaire comme étant l'ensemble des mots pouvant être dérivés depuis l'axiome.

Définition (Dérivation en une étape) : Soient une grammaire $G = (T, N, S, R)$, une forme non vide $u \in (N \cup T)^+$ et une forme éventuellement vide $v \in (N \cup T)^*$. La grammaire G permet de dériver v de u en une étape (noté $u \Rightarrow v$) si et seulement si :

— $u = xu'y$ (u peut être décomposé en x, u' et y ; x et y peuvent être vides),

— $v = xv'y$ (v peut être décomposé en x, v' et y),

— $u' \to v'$ est une règle de R.

Définition (Dérivation en plusieurs étapes) : Une forme v peut être dérivée d'une forme u en plusieurs étapes :

- $u \overset{+}{\Rightarrow} v$: si v peut être obtenue de u par une succession de 1 ou plusieurs dérivations en une étape,
- $u \overset{*}{\Rightarrow} v$: si v peut être obtenue de u par une succession de 0, 1 ou plusieurs dérivations en une étape.

Définition (Langage généré par une grammaire) : Le langage généré par une grammaire $G = (T, N, S, R)$ est l'ensemble des mots sur T qui peuvent être dérivés à partir de S :

$$\mathcal{L}(G) = \{v \in T^* / S \overset{+}{\Rightarrow} v\}$$

Remarques :
- Une grammaire définit un seul langage.
- Par contre, un même langage peut être engendré par plusieurs grammaires différentes.

Exercice : On considère la grammaire $G = (T, N, Ph, R)$ où

$$\begin{array}{rl} T = & \{ \ \ un, une, le, la, enfant, garcon, fille, cerise, haricot, cueille, mange \ \ \} \\ N = & \{ \ \ Ph, Gn, Gv, Df, Dm, Nf, Nm, V \ \ \} \\ R = & \{ \ \ Ph \rightarrow Gn\ Gv \\ & \quad Gn \rightarrow Df\ Nf \ \mid \ Dm\ Nm \\ & \quad Gv \rightarrow V\ Gn \\ & \quad Df \rightarrow une \ \mid \ la \\ & \quad Dm \rightarrow un \ \mid \ le \\ & \quad Nf \rightarrow fille \ \mid \ cerise \\ & \quad Nm \rightarrow enfant \ \mid \ garcon \ \mid \ haricot \\ & \quad V \rightarrow cueille \ \mid \ mange \ \ \} \end{array}$$

- La phrase *"une cerise cueille un enfant"* appartient-elle au langage $\mathcal{L}(G)$?
- Déterminer le nombre de phrases du langage décrit par G.

Correction :

— Pour montrer qu'une phrase appartient au langage, on construit une dérivation de l'axiome Ph jusqu'à la phrase. On souligne à chaque fois le symbole non terminal qui est remplacé par la dérivation.

$$\begin{array}{l} \underline{Ph} \Rightarrow \underline{Gn}\ Gv \Rightarrow Df\ Nf\ \underline{Gv} \Rightarrow Df\ Nf\ V\ \underline{Gn} \Rightarrow \underline{Df}\ Nf\ V\ Dm\ Nm \\ \quad \Rightarrow \text{une}\ \underline{Nf}\ V\ Dm\ Nm \Rightarrow \text{une cerise}\ \underline{V}\ Dm\ Nm \Rightarrow \text{une cerise cueille}\ \underline{Dm}\ Nm \\ \quad \Rightarrow \text{une cerise cueille un}\ \underline{Nm} \Rightarrow \text{une cerise cueille un enfant} \end{array}$$

Notons qu'il existe plusieurs dérivations possibles.

— Partant de l'axiome, on ne peut appliquer qu'une règle, qui dérive « Ph » en « $Gn\ Gv$ », et on ne peut appliquer qu'une seule règle pour ré-écrire Gv. Ainsi, l'ensemble des phrases que l'on peut générer à partir de Ph est égal à l'ensemble des phrases que l'on peut dériver à partir de « $Gn\ V\ Gn$ ». Chaque groupe nominal Gn peut être ré-écrit soit en « $Df\ Nf$ » soit en « $Dm\ Nm$ », et comme chaque non terminal Df, Nf, et Dm peut se ré-écrire en 2 terminaux différents tandis que Nm peut se ré-écrire en 3 terminaux différents, on peut générer $2 * 2 + 2 * 3 = 10$ suites de symboles terminaux différentes à partir de Gn. On peut par ailleurs ré-écrire V en 2 symboles terminaux, de sorte que le nombre total de phrases différentes que l'on peut générer à partir de Ph est égal à $10 * 2 * 10 = 200$.

Exercice : On considère la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \ \ b, c \ \ \} \\ N = & \{ \ \ S \ \ \} \\ R = & \{ \ \ S \rightarrow bS \ \mid \ cc \ \ \} \end{array}$$

Déterminer $\mathcal{L}(G)$.

Correction : $\mathcal{L}(G) = \{b^n cc / n \in \mathbb{N}\}$

En effet, partant de l'axiome S, toute dérivation commencera nécessairement par appliquer 0, 1 ou plusieurs fois la première règle puis se terminera en appliquant la deuxième règle. On représentera

cela en écrivant le schéma de dérivation suivant :

$$S \xrightarrow{n \text{ fois } (1)} b^n S \xrightarrow{(2)} b^n cc \quad \text{avec } n \in \mathbb{N}$$

Exercice : On considère la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \quad 0, 1 \ \} \\ N = & \{ \quad S \ \} \\ R = & \{ \quad S \to 0S \ | \ 1S \ | \ 0 \ \} \end{array}$$

Déterminer $\mathcal{L}(G)$.

Correction : $\mathcal{L}(G) = \{u0/u \in \{0,1\}^*\}$

En effet, partant de l'axiome S, toute dérivation commencera nécessairement par appliquer 0, 1 ou plusieurs fois la première ou la deuxième règle puis se terminera en appliquant la troisième règle. On représentera cela en écrivant le schéma de dérivation suivant :

$$S \xrightarrow{n \text{ fois } (1 \text{ ou } 2)} uS \xrightarrow{(3)} u0 \quad \text{avec } n \in \mathbb{N}, u \in \{0,1\}^* \text{ et } |u| = n$$

Exercice : On considère la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \quad a, b, 0 \ \} \\ N = & \{ \quad S, U \ \} \\ R = & \{ \quad S \to aSa \ | \ bSb \ | \ U \\ & \quad\ \ U \to 0U \ | \ \epsilon \ \} \end{array}$$

Déterminer $\mathcal{L}(G)$.

Correction : $\mathcal{L}(G) = \{u0^m v/u \in \{a,b\}^*, v = \text{inverse}(u), m \in \mathbb{N}\}$

où inverse(u) est le mot inverse de u, défini récursivement par :

- inverse$(\epsilon) = \{\epsilon\}$
- pour tout mot $u \in \mathcal{A}^+$ commençant par un symbole $a \in \mathcal{A}$ et se terminant par une suite de symboles $u' \in \mathcal{A}^*$ (tel que $u = a.u'$), inverse(u) = inverse$(u').a$

En effet, partant de l'axiome S, toute dérivation partant de S suivra nécessairement le schéma suivant :

$$S \xrightarrow{n \text{ fois } (1 \text{ ou } 2)} uSv \xrightarrow{(3)} uUv \xrightarrow{m \text{ fois } (4)} u0^m Uv \xrightarrow{(5)} u0^m v$$

avec $u \in \{a,b\}^*, v = \text{inverse}(u), n \in \mathbb{N}, |u| = n, m \in \mathbb{N}$

Exercice : Construire une grammaire pour le langage $\mathcal{L} = \{ab^n a/n \in \mathbb{N}\}$.

Correction : On définit la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \quad a, b \ \} \\ N = & \{ \quad S, U \ \} \\ R = & \{ \quad S \to aUa \\ & \quad\ \ U \to bU \ | \ \epsilon \ \} \end{array}$$

Exercice : Construire une grammaire pour le langage $\mathcal{L} = \{0^{2n}1^n / n \geq 0\}$.

Correction : On définit la grammaire $G = (T, N, S, R)$ où

$$\begin{aligned} T = & \quad \{ \quad 0, 1 \ \} \\ N = & \quad \{ \quad S \ \} \\ R = & \quad \{ \quad S \to 00S1 \ | \ \epsilon \ \} \end{aligned}$$

2.4 Types de grammaires

En introduisant des critères plus ou moins restrictifs sur la forme des règles de grammaire, on obtient des classes de grammaires hiérarchisées, ordonnées par inclusion. La classification des grammaires, définie en 1957 par Noam CHOMSKY, distingue les quatre classes suivantes :

Type 0 : pas de restriction sur les règles.

Type 1 : grammaires sensibles au contexte ou contextuelles. Les règles de R sont de la forme :

$$uAv \to uwv \text{ avec } A \in N,\ u, v \in (N \cup T)^* \text{ et } w \in (N \cup T)^+$$

Autrement dit, le symbole non terminal A est remplacé par w si on a les contextes u à gauche et v à droite.

Type 2 : grammaires hors-contexte. Les règles de R sont de la forme

$$A \to w \text{ avec } A \in N \text{ et } w \in (N \cup T)^*$$

Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal.

Type 3 : grammaires régulières

— à droite. Les règles de R sont de la forme

$$A \to aB \text{ ou } A \to a \text{ avec} A, B \in N \text{ et } a \in T$$

— à gauche. Les règles de R sont de la forme

$$A \to Ba \text{ ou } A \to a \text{ avec} A, B \in N \text{ et } a \in T$$

Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal, et le membre de droite est constitué d'un symbole terminal et éventuellement d'un symbole non terminal. Pour les grammaires régulières à droite, le symbole non terminal doit toujours se trouver à droite du symbole terminal tandis que pour les grammaires régulières à gauche il doit se trouver à gauche.

A chaque type de grammaire est associé un type de langage :

— les grammaires de type 3 génèrent les langages réguliers,
— les grammaires de type 2 génèrent les langages hors-contexte,
— les grammaires de type 1 génèrent les langages contextuels,
— les grammaires de type 0 permettent de générer tous les langages "décidables", autrement dit, tous les langages qui peuvent être reconnus en un temps fini par une machine. Les langages qui ne peuvent pas être générés par une grammaire de type 0 sont dits "indécidables".

Ces langages sont ordonnés par inclusion : l'ensemble des langages générés par les grammaires de type n est strictement inclus dans celui des grammaires de type $n-1$ (pour $n \in \{1, 2, 3\}$).

Enfin, à chaque type de grammaire est associé un type d'automate qui permet de reconnaître les langages de sa classe (c'est-à-dire de déterminer si un mot donné appartient au langage) : les langages réguliers sont reconnus par des automates finis, les langages hors-contexte sont reconnus par des automates à pile, et les autres langages, décrits par des grammaires de type 1 ou 0, sont reconnus par des machines de Turing. Ainsi, la machine de Turing peut être considérée comme le modèle de machine le plus puissant qu'il soit, dans la mesure où tout langage (ou plus généralement, tout problème) qui ne peut pas être traité par une machine de Turing, ne pourra pas être traité par une autre machine.

Exercice : On considère la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \quad a, b, c, d \ \} \\ N = & \{ \quad S, U \ \} \\ R = & \{ \quad S \to aU \ | \ c \\ & \quad\ \ U \to Sb \ | \ d \ \} \end{array}$$

Donner le type de G et déterminer $\mathcal{L}(G)$.

Correction : G est hors-contexte (car la partie gauche de chaque règle est un symbole non terminal). G n'est pas régulière car la règle (1) est régulière à droite tandis que la règle (3) est régulière à gauche. Notons qu'il n'existe pas de grammaire régulière permettant de générer $\mathcal{L}(G)$.

$\mathcal{L}(G) = \{a^n cb^n, a^{n+1} db^n / n \in \mathbb{N}\}$

En effet, partant de l'axiome S, toute dérivation partant de S suivra nécessairement un des deux schémas suivants :

$$S \xRightarrow{n \text{ fois (1 suivie de 3)}} a^n S b^n \xRightarrow{(2)} a^n c b^n$$
$$S \xRightarrow{n \text{ fois (1 suivie de 3)}} a^n S b^n \xRightarrow{(1)} a^n a U b^n \xRightarrow{(4)} a^n a d b^n$$

avec $n \in \mathbb{N}$

Exercice : On considère le langage $\mathcal{L}$ des mots sur $\{a, b, c\}$ qui contiennent au moins une fois la chaine *bac*. Définir formellement $\mathcal{L}$ et construire une grammaire hors-contexte puis une grammaire régulière décrivant $\mathcal{L}$.

Correction : $\mathcal{L} = \{u.bac.v / u \in \{a, b, c\}^*, v \in \{a, b, c\}^*\}$

On définit la grammaire hors-contexte $G = (T, N, S, R)$ où

$$\begin{array}{rl} T = & \{ \quad a, b, c \ \} \\ N = & \{ \quad S, U \ \} \\ R = & \{ \quad S \to UbacU \\ & \quad\ \ U \to Ua \ | \ Ub \ | \ Uc \ | \ \epsilon \ \} \end{array}$$

et la grammaire régulière à droite $G = (T, N, S, R)$ où

$$\begin{array}{rll} T = & \{ & a, b, c \ \} \\ N = & \{ & S, U, V, W \ \} \\ R = & \{ & S \to aS \mid bS \mid cS \mid bU \\ & & U \to aV \\ & & V \to cW \mid c \\ & & W \to aW \mid bW \mid cW \mid a \mid b \mid c \ \} \end{array}$$

ainsi que la grammaire régulière à gauche $G = (T, N, S, R)$ où

$$\begin{array}{rll} T = & \{ & a, b, c \ \} \\ N = & \{ & S, U, V, W \ \} \\ R = & \{ & S \to Sa \mid Sb \mid Sc \mid Uc \\ & & U \to Va \\ & & V \to Wb \mid b \\ & & W \to Wa \mid Wb \mid Wc \mid a \mid b \mid c \ \} \end{array}$$

Exercice : On considère le langage $\mathcal{L}$ des mots sur $\{0, 1\}$ qui représentent des entiers pairs non signés en base 2 (les mots de ce langage se terminent tous par 0 et ne commencent pas par 0, sauf pour l'entier nul). Définir formellement $\mathcal{L}$ et construire une grammaire régulière décrivant $\mathcal{L}$.

Correction : $\mathcal{L} = \{0, 1u0/u \in \{0, 1\}^*\}$

On définit la grammaire régulière à droite $G = (T, N, S, R)$ où

$$\begin{array}{rll} T = & \{ & a, b, c \ \} \\ N = & \{ & S, U \ \} \\ R = & \{ & S \to 0 \mid 1U \\ & & U \to 1U \mid 0U \mid 0 \ \} \end{array}$$

ainsi que la grammaire régulière à gauche $G = (T, N, S, R)$ où

$$\begin{array}{rll} T = & \{ & a, b, c \ \} \\ N = & \{ & S, U \ \} \\ R = & \{ & S \to 0 \mid U0 \\ & & U \to U1 \mid U0 \mid 1 \ \} \end{array}$$

Exercice : On considère la grammaire $G = (T, N, S, R)$ où

$$\begin{array}{rll} T = & \{ & a, b, c \ \} \\ N = & \{ & S, D, E \ \} \\ R = & \{ & S \to aSDE \mid \epsilon \\ & & aD \to ab \\ & & bE \to bc \\ & & cD \to DE \\ & & bD \to bb \\ & & cE \to cc \ \} \end{array}$$

- Quel est le type de G ?
- Ecrire la dérivation qui, partant de l'axiome, applique deux fois la première règle et une fois la seconde, et poursuivre la dérivation jusqu'à obtenir une chaine de terminaux.

— En raisonnant par récurrence, déterminer $\mathcal{L}(G)$.

Correction :

— G n'est pas hors-contexte car plusieurs règles comportent plusieurs symboles en partie gauche. G n'est pas contextuelle non plus car dans la règle $cD \to DE$, on ne retrouve pas le contexte gauche (c) de D en partie droite de la règle. Ainsi, G est de type 0.

— $S \overset{(1)}{\Longrightarrow} aSDE \overset{(1)}{\Longrightarrow} aaSDEDE \overset{(2)}{\Longrightarrow} aaDEDE \overset{(3)}{\Longrightarrow} aabEDE \overset{(4)}{\Longrightarrow} aabcDE$ $\overset{(5)}{\Longrightarrow} aabDEE \overset{(6)}{\Longrightarrow} aabbEE \overset{(4)}{\Longrightarrow} aabbcE \overset{(7)}{\Longrightarrow} aabbcc$

— $\mathcal{L}(G) = \{a^n b^n c^n / n \in \mathbb{N}\}$.
En effet, toute dérivation partant de S suivra nécessairement le schéma suivant :

$$S \xrightarrow{n \text{ fois } (1)} a^n S(DE)^n \overset{(2)}{\Longrightarrow} a^n (DE)^n \quad \text{avec } n \in \mathbb{N}$$

On vérifie facilement que $a^n(DE)^n = a^{n-1}aDE(DE)^{n-1}$ se dérive en $a^{n-1}abc(DE)^{n-1}$. Montrons maintenant par récurrence sur k que $b^k c^k DE$ se dérive en $b^{k+1}c^{k+1}, \forall k > 0$:

— Montrons cela pour $k = 1$:

$$bcDE \overset{(5)}{\Longrightarrow} bDEE \overset{(6)}{\Longrightarrow} bbEE \overset{(4)}{\Longrightarrow} bbcE \overset{(7)}{\Longrightarrow} bbcc$$

— Supposons que cela est vrai jusqu'au rang k :

$$b^k c^k DE \overset{Hyp}{\Longrightarrow} b^{k+1}c^{k+1}$$

— Montrons que cela est vrai au rang $k+1$:

$$b^{k+1}c^{k+1}DE \overset{(5)}{\Longrightarrow} bb^k c^k DEE \overset{Hyp}{\Longrightarrow} bb^{k+1}c^{k+1}E \overset{(7)}{\Longrightarrow} bb^{k+1}c^{k+1}c = b^{k+2}c^{k+2}$$

Par conséquent, en répétant ces dérivations $n-1$ fois de suite, on dérivera $a^{n-1}abc(DE)^{n-1}$ en $a^n b^n c^n$.

3 Langages réguliers et Automates finis

3.1 Grammaires régulières et langages réguliers

Définition (Grammaire régulière) : On rappelle qu'une grammaire $G = (T, N, S, R)$ est régulière

— à droite si les règles de R sont de la forme

$$A \to aB \text{ ou } A \to a \text{ avec} A, B \in N \text{ et } a \in T$$

— à gauche si les règles de R sont de la forme

$$A \to Ba \text{ ou } A \to a \text{ avec} A, B \in N \text{ et } a \in T$$

Exemple de grammaire régulière à droite : $G1 = (T_1, N_1, S_1, R_1)$ *avec*

$$\begin{aligned} T_1 = &\ \{ \ a, b \ \} \\ N_1 = &\ \{ \ S_1, U_1 \ \} \\ R_1 = &\ \{ \ S_1 \to aS_1 \mid aU_1 \\ &\quad\ U_1 \to bU_1 \mid b \ \} \end{aligned}$$

Exemple de grammaire régulière à gauche : $G2 = (T_2, N_2, S_2, R_2)$ *avec*

$$\begin{array}{rl} T_2 = & \{ \quad a, b \ \} \\ N_2 = & \{ \quad S_2, U_2 \ \} \\ R_2 = & \{ \quad S_2 \rightarrow S_2 b \ | \ U_2 b \\ & \quad\ U_2 \rightarrow U_2 a \ | \ a \quad \} \end{array}$$

G_1 *et* G_2 *engendrent le même langage :* $\mathcal{L}(G_1) = \mathcal{L}(G_2) = \{a^n b^p / n > 0, p > 0\}$

Définition (Langage régulier) : Un langage est régulier si et seulement s'il existe une grammaire régulière générant ce langage.

Les grammaires et langages réguliers sont la base de la lexicographie. L'ensemble des mots-clés, identificateurs, constantes numériques, ... d'un langage de programmation tel que le C++ est un langage régulier et peut être décrit par une grammaire régulière.

L'intérêt de distinguer grammaires régulières à droite ou à gauche apparait lors de l'analyse : si on lit les symboles du mot à analyser de la gauche vers la droite, alors
- une grammaire régulière à droite sera utilisée pour une analyse descendante, de l'axiome vers le mot ;
- une grammaire régulière à gauche sera utilisée pour une analyse ascendante, du mot vers l'axiome.

Par exemple, pour analyser le mot aaabb avec la grammaire G_1*, on construira la dérivation*

$$S_1 \Rightarrow aS_1 \Rightarrow aaS_1 \Rightarrow aaaU_1 \Rightarrow aaabU_1 \Rightarrow aaabb$$

tandis que pour analyser ce mot avec la grammaire G_2*, on construira la dérivation*

$$aaabb \Leftarrow U_2aabb \Leftarrow U_2abb \Leftarrow U_2bb \Leftarrow S_2b \Leftarrow S_2$$

3.2 Automates Finis Indéterministes

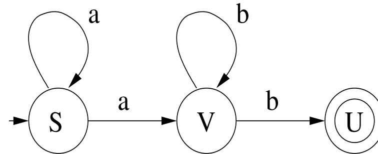
Un automate est une procédure effective (un algorithme) permettant de déterminer si un mot donné appartient à un langage. A la classe des langages réguliers correspond une classe particulière d'automates (reconnaissant les langages réguliers et seulement ceux-ci) : la classe des automates finis.

Définition (Automate Fini Indéterministe = AFI) : Un automate fini indéterministe est défini par un quintuplet (K, T, M, I, F) tel que

- K est un ensemble fini d'états.
- T est le vocabulaire terminal (correspondant à l'alphabet sur lequel est défini le langage).
- M est une relation dans $K \times T \times K$, appelée relation de transition (autrement dit, M est un ensemble de triplets de la forme (S_i, a, S_j) où S_i et S_j sont des états de K et a est un symbole du vocabulaire terminal T). Intuitivement, un triplet $(S_i, a, S_j) \in M$ signifie que si l'automate se trouve dans l'état S_i et le mot à analyser commence par le symbole a, alors l'automate peut aller dans l'état S_j.
- $I \subseteq K$ est l'ensemble des états initiaux.
- $F \subseteq K$ est l'ensemble des états finaux.

Représentation graphique d'un automate fini : On représente généralement un automate fini par un graphe orienté dont les arcs sont étiquetés. Chaque état de l'automate est représenté par un sommet du graphe. A chaque transition $(S_i, a, S_j) \in M$ on associe un arc du sommet S_i vers le sommet S_j étiqueté par a. Les sommets du graphe correspondant à des états initiaux de l'automate sont repérés par une pointe de flèche. Les sommets du graphe correspondant à des états finaux sont entourés de deux cercles.

Par exemple, l'AFI (K, T, M, I, F) *tel que*
- $K = \{S, V, U\}$,
- $T = \{a, b\}$,
- $M = \{(S, a, S), (S, a, V), (V, b, V), (V, b, U)\}$,
- $I = \{S\}$,
- $F = \{U\}$
sera représenté graphiquement par le graphe :



Fonctionnement d'un AFI : De façon informelle, un mot u est accepté par un AFI s'il existe un chemin d'un sommet initial vers un sommet final tel que la concaténation des étiquettes des arcs empruntés par le chemin soit égale à u. Sur l'exemple précédent, le langage des mots acceptés par l'automate est $\mathcal{L} = \{a^n b^p / n > 0, p > 0\}$.

De facon plus formelle, le fonctionnement d'un AFI $A = (K, T, M, I, F)$ est défini de la facon suivante :

— Une configuration de l'automate est caractérisée par un couple (S, u) tel que $S \in K$ est l'état courant et $u \in T*$ correspond à la fin du mot à analyser.
— La configuration (S', u') est dérivable en une étape de la configuration (S, u) (noté $(S, u) \Rightarrow (S', u')$) si $u = a.u'$ et $(S, a, S') \in M$.
— La configuration (S', u') est dérivable en plusieurs étapes de la configuration (S, u) (noté $(S, u) \overset{*}{\Rightarrow} (S', u')$) si (S', u') peut être obtenu de (S, u) par une succession de dérivations en une étape.
— Un mot w est accepté par l'automate s'il existe une dérivation

$$(S_0, w) \overset{*}{\Rightarrow} (S_i, \epsilon)$$

où $S_0 \in I$ est un état initial et $S_i \in F$ est un état final.
— Le langage $\mathcal{L}(A)$ accepté par un automate fini A est l'ensemble des mots acceptés par A.

Non déterminisme : Un tel automate est dit indéterministe car d'une part il peut y avoir plusieurs états initiaux, et d'autre part, étant donné un état $S_i \in K$ et un symbole $a \in T$, il peut exister plusieurs transitions possibles (au niveau de la représentation graphique par un graphe, ce non déterminisme correspond au cas où il y a plusieurs arcs étiquetés par un même symbole terminal qui partent du même sommet). Sur l'exemple précédent, quand l'automate se trouve dans l'état S et que le mot à analyser commence par a, il a le choix entre rester dans l'état S ou aller dans l'état T. Concrêtement, dans ce cas, l'automate choisit "au hasard" une des possibilités, et garde en mémoire le fait qu'il y a d'autres possibilités (on dit qu'il pose un point de choix). Si avec

la transition choisie il arrive à terminer la dérivation jusqu'à un état final, alors le mot est accepté et on arrête l'exécution. En revanche, si l'automate n'arrive pas à terminer la dérivation, alors il retourne jusqu'au dernier point de choix (on dit qu'il “backtrack") et recommence avec une autre possibilité.

3.3 Automates Finis Déterministes

L'exécution d'un automate fini indéterministe peut s'avérer très inefficace s'il comporte beaucoup de points de choix : si à chaque état l'automate a le choix entre deux transitions, alors pour analyser un mot de longueur n il faudra envisager, dans le pire des cas, de l'ordre de 2^n transitions (si on a de la chance, et que l'on choisit toujours la “bonne" dérivation en premier, on pourra cependant trouver une dérivation en n transitions). Pour éliminer ces points de choix, et rendre l'exécution efficace, il faut que l'automate soit déterministe, c'est-à-dire qu'il ait un seul état initial et que, étant donnés un état $S_i \in K$ et un symbole $a \in T$, il existe une seule transition possible.

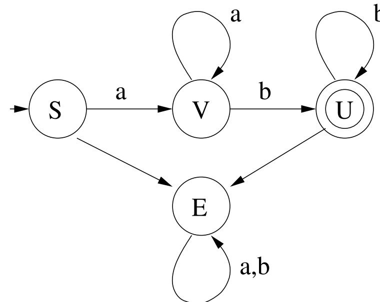
Définition (Automate Fini Déterministe = AFD) : Un automate fini déterministe est défini par un quintuplet (K, T, M, S_0, F) tel que

- K est un ensemble fini d'états.
- T est le vocabulaire terminal (correspondant à l'alphabet sur lequel est défini le langage).
- M est une fonction de $K \times T$ dans K, appelée fonction de transition ($M(S_i, a)$ donne l'état unique dans lequel l'automate doit aller quand il se trouve dans l'état S_i et que le mot à analyser commence par le symbole a).
- $S_0 \in K$ est l'état initial.
- $F \subseteq K$ est l'ensemble des états finaux.

Par exemple, l'AFD (K, T, M, S, F) tel que

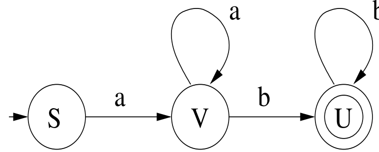
$$
\begin{array}{rlllll}
K & = \{ & S, V, U, E\} \\
T & = \{ & a, b\} \\
M & = \{ & (S,a) \rightarrow V & (S,b) \rightarrow E & (V,a) \rightarrow V & (V,b) \rightarrow U \\
 & & (U,a) \rightarrow E & (U,b) \rightarrow U & (E,a) \rightarrow E & (E,b) \rightarrow E \\
F & = \{ & U\}
\end{array}
$$

est représenté graphiquement par le graphe :



et accepte le langage $\mathcal{L} = \{a^n b^p / n > 0, p > 0\}$.
L'état E de cet automate correspond à un état d'erreur : l'automate va dans cet état dès lors qu'il reconnait que le mot ne fait pas partie du langage, et y reste jusque la fin

de l'analyse. Dans un souci de simplification, on ne représente généralement pas cet état, et on représente l'automate par le graphe



Un AFD fonctionne comme un AFI, et on définit de la même manière les notions de configuration, dérivation entre configurations et acceptation d'un mot, la seule différence étant que la fonction de transition M détermine de façon unique le nouvel état dans lequel l'automate doit se placer au moment de faire une dérivation. L'exécution d'un automate fini déterministe est résumée dans la procédure "accepte" suivante :

procédure accepte
entrée : un AFD (K, T, M, S_0, F)
  un tableau de caractères u indicé de 1 à n
sortie : retourne vrai si $u[1..n]$ appartient au langage, faux sinon
debut
  $etatCrt \leftarrow S_0$
  $i \leftarrow 1$
  tant que $i \leq n$ faire
    $etatCrt \leftarrow M(etatCrt, u[i])$
    $i \leftarrow i + 1$
  fin tant que
  si $etatCrt \in F$ alors retourne vrai sinon retourne faux
fin

3.4 Equivalence entre AFI et AFD

Pour déterminer si un mot u de longueur n est accepté, un AFD effectue exactement n transitions, tandis qu'un AFI en effectue de l'ordre de 2^n. L'exécution d'un AFD est donc nettement plus efficace que celle d'un AFI. En contrepartie, on peut se demander si les AFI sont plus généraux, c'est-à-dire s'ils acceptent plus de langages que les AFD. La réponse, négative, est donnée par le théorème suivant.

Théorème (équivalence entre AFD et AFI) : La famille des langages acceptés par un AFD est identique à la famille des langages acceptés par un AFI (autrement dit, s'il existe un AFI reconnaissant un langage donné, alors il existe un AFD reconnaissant le même langage).

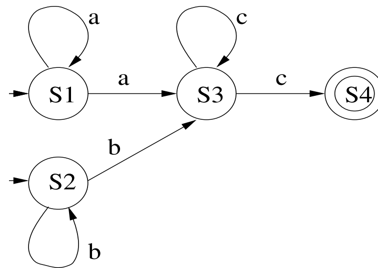
La démonstration de ce théorème est réalisée en donnant un algorithme permettant de construire à partir d'un AFI un AFD reconnaissant le même langage. Chaque état de l'AFD correspond à un ensemble d'états de l'AFI.

procédure rendDéterministe
entrée : un AFI $A = (K, T, M, I, F)$
sortie : un AFD $A' = (K', T, M', S'_0, F')$ tel que $\mathcal{L}(A) = \mathcal{L}(A')$
début
    $S'_0 \leftarrow I$
    $K' \leftarrow \{I\}$
    $vus \leftarrow \emptyset$
    tant que $K' \neq vus$ faire
        soit $\mathcal{U}$ un état de K' tel que $\mathcal{U} \notin vus$
        pour tout $l \in T$ faire
            $\mathcal{V} \leftarrow \{S_j / \exists S_i \in \mathcal{U}, (S_i, l, S_j) \in M\}$
            $M'(\mathcal{U}, l) \leftarrow \mathcal{V}$
            $K' \leftarrow K' \cup \{\mathcal{V}\}$
        fin pour
        $vus \leftarrow vus \cup \{\mathcal{U}\}$
    fin tant que
    $F' \leftarrow \{\mathcal{U} \in K' / \mathcal{U} \cap F = \emptyset\}$
fin

Considérons par exemple l'AFI $(K, T, M, I, F))$ suivant :

$$\begin{aligned} K = & \{S_1, S_2, S_3, S_4\} \\ T = & \{a, b, c\} \\ M = & \{(S1, a, S_1), (S_1, a, S_3), (S2, b, S_2), (S_2, b, S_3), (S_3, c, S_3), (S_3, c, S_4)\} \\ I = & \{S_1, S_2\} \\ F = & \{S_4\} \end{aligned}$$

correspondant au graphe suivant :



Pour plus de commodités, on représente la relation de transition M par la table suivante :

	a	b	c
S_1	$\{S_1, S_3\}$	$\emptyset$	$\emptyset$
S_2	$\emptyset$	$\{S_2, S_3\}$	$\emptyset$
S_3	$\emptyset$	$\emptyset$	$\{S_3, S_4\}$
S_4	$\emptyset$	$\emptyset$	$\emptyset$

On construit ensuite les états de l'AFD et leur fonction de transition. Au départ, l'AFD a un seul état qui est composé de l'ensemble des états initiaux de l'AFI : sur notre exemple, l'état initial de l'AFD est $\{S_1, S_2\}$. A chaque fois qu'on ajoute un nouvel état dans l'AFD, on détermine sa

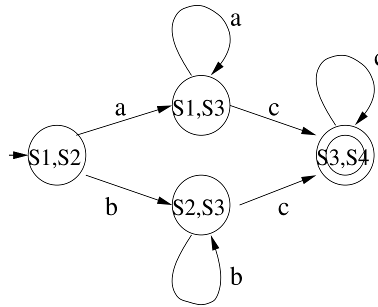
fonction de transition en faisant l'union des lignes correspondantes dans la table de transition de l'AFI : sur notre exemple, pour l'état $\{S_1, S_2\}$, on fait l'union des lignes correspondant à S_1 et S_2, et on détermine la fonction de transition

M	a	b	c
$\{S_1, S_2\}$	$\{S_1, S_3\}$	$\{S_2, S_3\}$	$\emptyset$

Autrement dit, quand on est dans l'état "S_1 ou S_2" et qu'on lit un a, on va dans l'état "S_1 ou S_3" ($M(\{S_1, S_2\}, a) = \{S_1, S_3\}$), quand on est dans l'état "S_1 ou S_2" et qu'on lit un b, on va dans l'état "S_2 ou S_3" ($M(\{S_1, S_2\}, b) = \{S_2, S_3\}$) et quand on est dans l'état "S_1 ou S_2" et qu'on lit un c, on va dans l'état "vide", correspondant à l'état d'erreur ($M(\{S_1, S_2\}, c) = \emptyset$. On rajoute ensuite les états $\{S_1, S_3\}$ et $\{S_2, S_3\}$ à l'AFD et on détermine leur fonction de transition selon le même principe. De proche en proche, on construit la table de transition suivante pour l'AFD :

	a	b	c
$\{S_1, S_2\}$	$\{S_1, S_3\}$	$\{S_2, S_3\}$	$\emptyset$
$\{S_1, S_3\}$	$\{S_1, S_3\}$	$\emptyset$	$\{S_3, S_4\}$
$\{S_2, S_3\}$	$\emptyset$	$\{S_2, S_3\}$	$\{S_3, S_4\}$
$\{S_3, S_4\}$	$\emptyset$	$\emptyset$	$\{S_3, S_4\}$

L'ensemble des états de l'AFD est $K' = \{\{S_1, S_2\}, \{S_1, S_3\}, \{S_2, S_3\}, \{S_3, S_4\}\}$. Les états de l'AFD contenant un état final de l'AFI sont des états finaux. Ici, l'AFI a un seul état final S_4 et l'ensemble des états finaux de l'AFD est $F' = \{\{S_3, S_4\}\}$. Cet AFD correspond au graphe suivant :



3.5 Equivalence entre automates finis et langages réguliers

On a défini au début de ce chapitre les langages réguliers comme étant les langages que l'on peut décrire par une grammaire régulière. On a ensuite décrit les automates finis et on a montré l'équivalence entre automates finis déterministes et indéterministes. On va montrer maintenant l'équivalence entre langages réguliers et automates finis.

Théorème : Tout langage accepté par un automate fini est régulier.

Pour démontrer ce théorème, on montre que pour tout automate fini déterministe $A = (K, T, M, S_0, F)$, il existe une grammaire régulière G telle que $\mathcal{L}(A) = \mathcal{L}(G)$. En effet, on construit la grammaire régulière à droite $G = (T, K, S_0, R)$ telle que

$R = \{U \leftarrow aV \ / \ U \in K, a \in T \text{ et } V = M(U, a)\} \cup \{U \leftarrow a \ / \ U \in K, a \in T, V = M(U, a) \text{ et } V \in F\}$

Théorème : Tout langage régulier est accepté par un automate fini.

Pour démontrer ce théorème, on montre que pour toute grammaire régulière à droite $G = (T, N, S_0, R)$, il existe un automate fini indéterministe A tel que $\mathcal{L}(A) = \mathcal{L}(G)$. En effet, on construit l'AFI $A = (K, T, M, I, F)$ tel que
- $K = N \cup \{S_f\}$ (où S_f est un nouveau symbole n'apparaissant pas dans N)
- $M = \{(A, l, B)$ / "$A \to lB$" $\in R\} \cup \{(A, l, S_f)/$"$A \to l$" $\in R\}$
- $I = \{S_0\}$
- $F = \{S_f\}$
De la même façon, on montre que pour toute grammaire régulière à gauche $G = (T, N, S_0, R)$, il existe un automate fini indéterministe A tel que $\mathcal{L}(A) = \mathcal{L}(G)$. En effet, on construit l'AFI $A = (K, T, M, I, F)$ tel que
- $K = N \cup \{S_i\}$ (où S_i est un nouveau symbole n'apparaissant pas dans N)
- $M = \{(B, l, A)$ / "$A \to Bl$" $\in R\} \cup \{(S_i, l, A)/$"$A \to l$" $\in R\}$
- $I = \{S_i\}$
- $F = \{S_0\}$

3.6 Expressions régulières

Les expressions régulières permettent de décrire les langages réguliers, de façon plus simple qu'en utilisant des opérations ensemblistes.

Définition (expression régulière) : Une expression E est une expression régulière sur un alphabet $\mathcal{A}$ si et seulement si
— $E = \emptyset$ ou
— $E = \epsilon$ ou
— $E = a$ avec $a \in \mathcal{A}$ ou
— $E = E_1 \mid E_2$ et E_1 et E_2 sont deux expressions régulières sur $\mathcal{A}$ ou
— $E = E_1.E_2$ et E_1 et E_2 sont deux expressions régulières sur $\mathcal{A}$ ou
— $E = E_1^*$ et E_1 est une expression régulière sur $\mathcal{A}$
Les opérateurs $*$, . et | ont une priorité décroissante. Si nécessaire, on peut ajouter des parenthèses.

Définition (langage décrit par une expression régulière) : le langage $\mathcal{L}(E)$ décrit par une expression régulière E définie sur un alphabet $\mathcal{A}$ est défini par
— $\mathcal{L}(E) = \emptyset$ si $E = \emptyset$,
— $\mathcal{L}(E) = \{\epsilon\}$ si $E = \epsilon$,
— $\mathcal{L}(E) = \{a\}$ si $E = a$,
— $\mathcal{L}(E) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$ si $E = E_1 \mid E_2$,
— $\mathcal{L}(E) = \mathcal{L}(E_1).\mathcal{L}(E_2)$ si $E = E_1.E_2$,
— $\mathcal{L}(E) = \mathcal{L}(E_1)^*$ si $E = E_1^*$ où E_1 est une expression régulière sur $\mathcal{A}$.

Par exemple, sur l'alphabet $\mathcal{A} = \{a, b, c\}$,
- *$E1 = a^*bbc^*$ décrit le langage $\mathcal{L}(E1) = \{a^n bbc^p / n \geq 0, p \geq 0\}$*
- *$E2 = (a \mid b \mid c)^*(bb \mid cc)a^*$ décrit le langage $\mathcal{L}(E2) = \{wbba^n, wcca^n / w \in \mathcal{A}^*, n \geq 0\}$*

L'équivalence entre expressions régulières et langages réguliers est établie par les deux théorèmes suivants.

Théorème : Toute expression régulière décrit un langage régulier.

Pour démontrer ce théorème, on montre que étant donnée une expression régulière E définie sur un alphabet T, on peut construire un automate fini indéterministe $A = (K, T, M, I, F)$ tel que $\mathcal{L}(E) = \mathcal{L}(A)$:

- si $E = \emptyset$ alors $K = \{S_0, S_f\}$, $M = \emptyset$, $I = \{S_0\}$ et $F = \{S_f\}$.
- si $E = \epsilon$ alors $K = \{S_0\}$, $M = \emptyset$, $I = \{S_0\}$ et $F = \{S_0\}$
- si $E = a$ alors $K = \{S_0, S_f\}$, $M = \{(S_0, a, S_f)$, $I = \{S_0\}$ et $F = \{S_f\}$
- si $E = E_1 \mid E_2$ alors on construit récursivement deux AFI $A_1 = (K_1, T, M_1, I_1, F_1)$ et $A_2 = (K_2, T, M_2, I_2, F_2)$ reconnaissant respectivement $\mathcal{L}(E_1)$ et $\mathcal{L}(E_2)$, et on construit A à partir de A_1 et A_2 :
  - $K = K_1 \cup K_2 \cup \{S_i, S_f\}$,
  - $M = M_1 \cup M_2 \cup \{(S_i, \epsilon, S)/S \in I_1 \cup I_2\} \cup \{(S, \epsilon, S_f)/S \in F_1 \cup F_2\}$
  - $I = \{S_i\}$
  - $F = \{S_f\}$.
- si $E = E_1.E_2$ alors on construit récursivement deux AFI $A_1 = (K_1, T, M_1, I_1, F_1)$ et $A_2 = (K_2, T, M_2, I_2, F_2)$ reconnaissant respectivement $\mathcal{L}(E_1)$ et $\mathcal{L}(E_2)$, et on construit A à partir de A_1 et A_2 :
  - $K = K_1 \cup K_2$
  - $M = M_1 \cup M_2 \cup \{(S_f, \epsilon, S_i)/S_f \in F_1, S_i \in I_2\}$
  - $I = I_1$
  - $F = F_2$.
- si $E = E_1^*$ alors on construit récursivement un AFI $A_1 = (K_1, T, M_1, I_1, F_1)$ reconnaissant $\mathcal{L}(E_1)$, et on construit A à partir de A_1 :
  - $K = K_1 \cup \{S_0\}$,
  - $M = M_1 \cup \{(S_0, \epsilon, S_i)/S_i \in I_1\} \cup \{(S_f, \epsilon, S_0)/S_f \in F_1\}$
  - $I = \{S_0\}$
  - $F = \{S_0\}$.

L'AFI ainsi construit contiendra des transitions sur ϵ. Ces transitions peuvent toujours être supprimées (mais on ne verra pas ici l'algorithme permettant de supprimer ces transitions sur ϵ).

Théorème : Tout langage régulier peut être décrit par une expression régulière.

Pour démontrer ce théorème, on peut montrer comment construire une expression régulière à partir d'un automate fini déterministe. L'idée est de décrire tous les chemins entre l'état initial et un état final, les boucles étant traitées par l'introduction de l'opérateur $*$. L'expression régulière finale est alors l'union des expressions régulières ainsi obtenues. On ne développera pas plus ici la façon de construire ces expressions régulières.

3.7 Quelques propriétés des langages réguliers

Théorème : Soient $\mathcal{L}_1$ et $\mathcal{L}_2$ deux langages réguliers. Les langages $\mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1.\mathcal{L}_2$, $c(\mathcal{L}_1)$ et $\mathcal{L}_1^*$ sont des langages réguliers.

Pour démontrer ce théorème, on peut montrer comment construire un AFD reconnaissant ces langages à partir des AFD reconnaissant $\mathcal{L}_1$ et $\mathcal{L}_2$. Pour l'union, la concaténation et l'itéré, la construction est évidente et a été vue au niveau des expressions régulières. Pour le complémentaire, il suffit d'échanger les états finaux et les autres états de l'automate. Pour l'intersection, on peut

par exemple utiliser le fait que $\mathcal{L}_1 \cap \mathcal{L}_2 = c(c(\mathcal{L}_1) \cup c(\mathcal{L}_2))$.

4 Langages hors-contexte et Automates à pile

Certains langages ne peuvent pas être décrits par une grammaire régulière, et ne peuvent donc pas être reconnus par un automate fini (par exemple le langage $\{a^n b^n / n > 0\}$). On étudie dans ce chapitre une classe de langages plus générale que celle des langages réguliers : la classe des langages hors-contexte, décrits par des grammaires hors-contexte et reconnus par des automates à pile.

Définition (Grammaire hors-contexte) : : $G = (T, N, S, R)$ est une grammaire hors-contexte si toutes les règles de R sont de la forme $A \rightarrow w$ avec $A \in N$ et $w \in (N \cup T)*$.

Définition (Langage hors-contexte) : On appelle langage hors-contexte un langage généré par une grammaire hors contexte.

4.1 Arbres syntaxiques

Dans le cas d'une grammaire hors-contexte G, on peut représenter la dérivation d'une phrase de $L(G)$ à partir de l'axiome à l'aide d'un arbre syntaxique (ou arbre d'analyse ou arbre de dérivation). Cette représentation fait abstraction de l'ordre d'application des règles de la grammaire et aide à la compréhension de la syntaxe de la phrase considérée.

Définition (Arbre syntaxique) : L'arbre syntaxique d'une phrase relativement à la grammaire hors-contexte $G = (T, N, S, R)$ est un arbre tel que

- la racine est étiquetée par le symbole de départ S,
- chaque noeud interne est étiqueté par un symbole non terminal de N,
- chaque feuille est étiquetée par un symbole terminal de T,
- pour tout noeud interne, si son étiquette est le symbole non terminal A, et si ses n fils ont respectivement pour étiquettes X_1, X_2, ..., X_n, alors

$$A \rightarrow X_1 X_2 ... X_n$$

  doit être une règle de R.

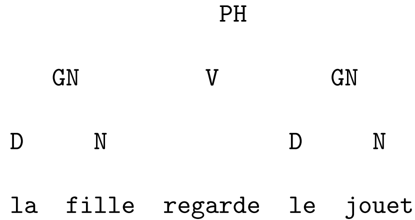
Remarques :

- La lecture de gauche à droite des feuilles de l'arbre reconstitue la phrase que l'arbre représente.
- Etant donnée une grammaire hors-contexte G, une phrase w est générée par G si et seulement si il existe un arbre syntaxique pour G qui génère w.

Considérons par exemple la grammaire G définie en 1.3 et dont les règles sont :

$PH \rightarrow GN\ V\ GN$	$N \rightarrow fille$
$GN \rightarrow D\ N$	$N \rightarrow jouet$
$D \rightarrow le$	$V \rightarrow regarde$
$D \rightarrow la$	

L'arbre syntaxique associé au mot *la fille regarde le jouet* est :



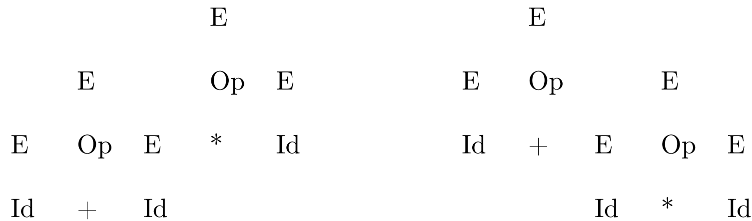
Définition (Phrase ambigüe) : Une phrase, générée par une grammaire, est ambigüe si elle admet plus d'un arbre syntaxique pour cette grammaire.

Définition (Grammaire ambigüe) : Une grammaire est ambigüe si elle génère au moins une phrase ambigue.

Considérons par exemple la grammaire $G = (T, N, E, R)$ avec :

T = { Id, +, -, *, /, (,) } où Id signifie identificateur
N = { E, Op }
R = { E → Id | (E) | E Op E
Op → + | - | * | / }

et considérons les deux arbres suivants :



Dans les deux cas, la phrase associée est "Id + Id * Id". On a donc deux arbres de dérivation pour une même phrase, et la grammaire G est ambigüe. En l'occurrence, on ne peut pas savoir si l'on commence par l'addition ((Id+Id)*Id) ou la multiplication (Id+(Id*Id)).

Remarques :

— Le terme "ambigu" est appliqué à la grammaire et non au langage : il est souvent possible de transformer une grammaire ambigüe en une grammaire non ambigüe générant le même langage. Cependant, il existe des langages pour lesquels il n'existe pas de grammaire non ambigüe ; de tels langages sont dits intrinsèquement ambigus.

— La propriété d'ambigüité est indécidable : cela signifie qu'il n'existe pas – et ne peut pas exister – d'algorithme général qui, étant donnée une grammaire hors-contexte, puisse déterminer en un temps fini si la grammaire est ambigüe ou non. Seules peuvent être déterminées des conditions suffisantes assurant la non-ambigüité.

4.2 La forme de BACKUS-NAUR d'une grammaire

La notation de Backus-Naur (en anglais Backus-Naur Form, ou BNF) a été utilisée dès 1960 pour décrire le langage ALGOL 60, et depuis est employée pour définir de nombreux langages de programmation. L'écriture BNF des règles de grammaire est définie comme suit :

- Le symbole $\rightarrow$ des règles de réécriture est remplacé par ::=,
- Les symboles désignant les éléments non-terminaux sont inclus entre chevrons $<$ et $>$, ceci afin de les distinguer des terminaux,
- Un ensemble de règles dont les parties gauches sont identiques, telles que

  $$A ::= u1, A ::= u2, ...A ::= up$$

  peut être écrit de manière abrégée :

  $$A ::= u1|u2|...|up$$

4.3 Propriétés de fermeture des langages hors-contexte

On rappelle qu'un ensemble est fermé relativement à une opération à n opérandes si tout résultat de cette opération appliquée à n éléments de l'ensemble appartient encore à l'ensemble.

Théorème : l'union de deux langages hors-contexte est un langage hors-contexte,
le produit de deux langages hors-contexte est un langage hors-contexte,
et l'itéré d'un langage hors contexte est un langage hors contexte.

En effet, soient deux langages :
- $\mathcal{L}_1$ généré par la grammaire $G_1 = (T_1, N_1, S_1, R_1)$ et
- $\mathcal{L}_2$ généré par la grammaire $G_2 = (T_2, N_2, S_2, R_2)$,
tels que $N_1 \cap N_2 = \emptyset$ (si tel n'est pas le cas, on renomme certains éléments non-terminaux) alors

- Le langage $\mathcal{L}_1 \cup \mathcal{L}_2$ est généré par la grammaire hors contexte $G = (T, N, S, R)$ telle que
  - $T = T_1 \cup T_2$,
  - S est un nouveau symbole non terminal ($S \notin N_1 \cup N_2$),
  - $N = N_1 \cup N_2 \cup \{S\}$,
  - $R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$.
- Le langage $\mathcal{L}_1\mathcal{L}_2$ est généré par la grammaire hors contexte $G = (T, N, S, R)$ telle que
  - $T = T_1 \cup T_2$,
  - S est un nouveau symbole non terminal ($S \notin N_1 \cup N_2$),
  - $N = N_1 \cup N_2 \cup \{S\}$,
  - $R = R_1 \cup R_2 \cup \{S \rightarrow S_1S_2\}$.
- Le langage $\mathcal{L}_1*$ est généré par la grammaire hors contexte $G = (T_1, N, S, R)$ telle que
  - S est un nouveau symbole non terminal ($S \notin N_1 \cup N_2$),
  - $N = N_1 \cup \{S\}$,
  - $R = R_1 \cup \{S \rightarrow S_1S, S \rightarrow \epsilon\}$.

En revanche, l'intersection de deux langages hors-contexte et le complémentaire d'un langage hors-contexte ne sont pas nécessairement des langages hors contexte.

4.4 Automates à pile

Les langages hors-contexte, décrits par des grammaires hors-contexte, sont reconnus (acceptés) par des automates à pile. De facon informelle, un automate à pile est un automate fini auquel on a ajouté une pile de capacité illimitée initialement vide. L'exécution d'un automate à pile sur un mot donné est semblable à celle d'un automate fini. Toutefois, à chaque étape, l'automate à pile consulte le sommet de sa pile et le remplace éventuellement par une suite de symboles.

Définition (Automate à pile) : Un automate à pile est un quintuplet $A = (T, P, Q, M, S_0)$ tel que

- T est le vocabulaire terminal,
- P est le vocabulaire de pile (contenant en particulier un symbole initial de pile vide, noté $\perp$),
- Q est un ensemble fini d'états,
- M est un ensemble de transitions,
- $S_0 \in Q$ est l'état initial.

Le vocabulaire de pile contient l'ensemble des symboles qui pourront apparaître sur la pile, et n'est pas nécessairement distinct du vocabulaire terminal (on peut avoir $T \cap P \neq \emptyset$).

Une transition de l'automate à pile est semblable à celle d'un automate fini, à part qu'elle spécifie en plus la manipulation de la pile. Une transition de M est de la forme

`(état, symbole_lu, sommet_pile)` $\rightarrow$ `(nouvel_état, action_sur_pile)`

tel que

- `état` et `nouvel_état` sont des états de Q,
- `symbole_lu` est soit un symbole terminal de T, soit ϵ, signifiant que l'automate ne regarde pas le mot,
- `sommet_pile` est soit un symbole de P, soit ϵ, signifiant que l'automate ne regarde pas le sommet de la pile, et
- `action_sur_pile` est soit :

  "dépiler le symbole au sommet de la pile", ou
  "dépiler le symbole au sommet de la pile puis empiler une suite de symboles" ou
  "empiler une suite de symboles", ou
  "ne rien faire".

De façon informelle, la transition

$$(S_i, u, v) \rightarrow (S_k, \texttt{action_sur_pile})$$

signifie que l'automate peut passer de l'état S_i à l'état S_k, pour autant que le mot à analyser commence par le symbole u et que la pile ait en sommet le symbole v. Après la transition, l'automate a consommé le symbole u du mot à analyser et a effectué l'action spécifiée dans âction_sur_pile. Un triplet (S_i, u, v) est appelé une **situation**.

De facon plus formelle, le fonctionnement de l'automate $A = (T, P, Q, M, S_0)$ est défini de la facon suivante :

- Une **configuration** de l'automate est caractérisée par un triplet :

$$(S, m, p) \text{ avec } S \in Q, m \in T^* \text{ et } p \in P^*$$

c'est à dire, un état courant S, une suite m de symboles terminaux (correspondant à la fin du mot à analyser) et une suite p de symboles de pile (correspondant à l'état courant de la pile).

— La configuration (S', m', p') est **dérivable en une étape** de la configuration (S, m, p) (noté $(S, m, p) \Rightarrow (S', m', p')$) si
  — $(S, u, v) \rightarrow (S', \texttt{action_sur_pile})$ est une transition de M,
  — $m = u.m'$ (le mot m à analyser commence par le symbole u),
  — le symbole au sommet de la pile p est v.
  — p' est la pile obtenue après avoir effectué l'action de `action_sur_pile` sur la pile p.
— La configuration (S', m', p') est **dérivable en plusieurs étapes** de la configuration (S, m, p) (noté $(S, m, p) \Rightarrow^* (S', m', p')$) si (S', m', p') peut être obtenu de (S, m, p) par une succession de dérivations en une étape.
— Un mot w est **accepté par l'automate à pile** A si

$$(S_0, w, \perp) \Rightarrow^* (S, \epsilon, \perp)$$

où $\perp$ est le symbole de pile vide.
Autrement dit, un mot est accepté par l'automate s'il existe une suite de transitions à partir de l'état initial et pile vide, conduisant à la lecture entière du mot et à la pile à nouveau vide.
— Le langage $\mathcal{L}(A)$ accepté par l'automate à pile A est l'ensemble des mots acceptés par A.

Les automates à pile que nous avons définis acceptent un mot lorsqu'il existe une dérivation menant à une configuration où la pile est vide et le mot entièrement lu. Pour cette raison, ils sont appelés automates à pile acceptant sur pile vide. Une autre définition des automates à pile est celle des automates à pile acceptant sur état final. Un tel automate spécifie en plus un ensemble $F \subseteq Q$ d'états finaux. Un mot est accepté s'il existe une dérivation menant à une configuration où l'état est final (la pile n'étant pas nécessairement vide) :

$$(S_0, w, \perp) \Rightarrow^* (S, \epsilon, p) \text{ avec } S \in F$$

Exemple

Soit l'automate à pile reconnaissant le langage des mots formés sur $\{a, b\}^*$ et contenant le même nombre de a et de b : $A = (T, P, Q, M, q)$ tel que
- $T = \{a, b\}$
- $P = \{a, b\}$
- $Q = \{q\}$
- $M =$

Etat	Symbole lu	Sommet de pile	Nouvel etat	Action sur pile
q	a	$\perp$	q	empiler(a)
q	b	$\perp$	q	empiler(b)
q	a	a	q	empiler(a)
q	b	b	q	empiler(b)
q	a	b	q	depiler(b)
q	b	a	q	depiler(a)

Le fonctionnement de cet automate sur le mot $w = aabbabba$ est :

Etat	Mot	Pile
q	$aabbabba$	$\perp$
q	$abbabba$	$a\perp$
q	$bbabba$	$aa\perp$
q	$babba$	$a\perp$
q	$abba$	$\perp$
q	bba	$a\perp$
q	ba	$\perp$
q	a	$b\perp$
q	ϵ	$\perp$

Comme pour les automates finis, on peut donner des automates à pile une représentation par graphe.

4.5 Automates à pile déterministes

Définition (Automate à pile déterministe) : Un automate à pile est dit déterministe si à toute situation ou configuration donnée ne correspond au plus qu'une transition.

Dans le cas d'un automate déterministe, si un mot est accepté par le langage, alors il existe une seule dérivation possible pour ce mot. En revanche, si un automate n'est pas déterministe, il peut y avoir des configurations pour lesquelles plusieurs transitions différentes sont possibles. Certaines de ces transitions peuvent amener à des échecs (l'automate est bloqué dans une configuration qui n'est pas d'acceptation) tandis que d'autres peuvent amener à une dérivation succès. Dans ce cas, il est nécessaire d'examiner toutes les transitions possibles jusqu'à en trouver une qui réussisse. Par conséquent, le temps nécessaire à un automate non déterministe pour reconnaître un mot est généralement bien plus long que pour un automate déterministe.

Une question naturelle est de se demander si tout langage hors-contexte peut-être accepté par un automate à pile déterministe. La réponse à cette question est malheureusement non : il existe des langages hors-contexte qui ne sont acceptés par aucun automate à pile déterministe. Les langages acceptés par les automates à piles déterministes forment donc une sous-classe des langages hors-contexte : la classe des langages hors-contexte déterministes.

L'application principale des langages hors-contexte est la description de la syntaxe des langages de programmation. Pour obtenir des compilateurs efficaces (pouvant être appliqués à de très longs programmes), il est intéressant de se limiter à des grammaires décrivant des langages hors-contexte déterministes.

4.6 Automates à pile et langages hors-contexte

Théorème : Un langage est hors-contexte si et seulement si il est accepté par un automate à pile.

En effet, à partir d'une grammaire hors-contexte $G = (T, N, S, R)$, on construit l'automate à pile $A = (T', P', Q', M', S')$ reconnaissant le même langage de la facon suivante :

- $T' = T$
- $P' = T \cup N \cup \{\perp\}$
- $Q' = \{p, q\}$ tel que p et q ne sont pas des symboles de $T \cup N$
- $$\begin{array}{rcl} M' & = & \{(p, \epsilon, \perp) \rightarrow (q, [\text{empiler(S)}])\} \\ & \cup & \{(q, \epsilon, B) \rightarrow (q, [\text{depiler(B)}, \text{empiler(S}_\text{n}\text{)}, \ldots, \text{empiler(S}_2\text{)}, \text{empiler(S}_1\text{)}])/ \\ & & (B \rightarrow S_1 S_2 \ldots S_n) \in R\} \\ & \cup & \{(q, a, a) \rightarrow (q, [\text{depiler(a)}])/a \in T\} \end{array}$$
- $S' = p$

Intuitivement, l'automate fonctionne de la facon suivante :
- Si le symbole en sommet de pile est un symbole non terminal B, on le remplace par une forme u telle qu'il existe une règle $B \rightarrow u$ dans R.
- Si le symbole en sommet de pile est un symbole terminal a, et le symbole en tête de la chaine à analyser est a, alors on dépile a de la pile, et on enlève a du mot à analyser.

Il est à noter qu'un automate construit de cette facon sera généralement non déterministe : dès qu'il existe plusieurs règles ayant un même symbole B en partie gauche, il y a plusieurs transitions dont la partie gauche est (q, ϵ, B).