PUBLICATIONS DU DÉPARTEMENT DE MATHÉMATIQUES DE LYON

SERGE GRIGORIEFF

Introduction à la théorie des langages

Publications du Département de Mathématiques de Lyon, 1982, fascicule 1B
« Quelques thèmes de la théorie des algorithmes », , p. 37-65

<http://www.numdam.org/item?id=PDML_1982___1B_37_0>

INTRODUCTION A LA THEORIE DES LANGAGES

par

Serge GRIGORIEFF

(Université de LYON I)

§ 1 - GRAMMAIRES ET LANGAGES RECURSIVEMENT ENUMERABLES.

1.1. Considérons une théorie mathématique (comme l'arithmétique de PEANO ou la théorie des ensembles) ; c'est la donnée

- d'un alphabet comportant des symboles logiques et des symboles propres à la théorie,

- de règles de formation des termes, formules et -parmi ces dernières- des théorèmes de la théorie.

Ces termes, formules et théorèmes sont des ensembles de mots construits sur un même alphabet : nous dirons que ce sont des langages.

Examinons quelques caractères du plus compliqué de ces langages, l'ensemble des théorèmes :

(i) - Il n'est pas toujours possible d'avoir un algorithme pour décider si une formule est ou non un théorème (c'est par exemple hors de question pour l'arithmétique de PEANO, mais possible pour l'arithmétique purement additive).

(ii) - En revanche, il est facile de savoir si une suite de formules est ou non une preuve (à supposer que l'on puisse décider si une formule est ou non un axiome, ce qui est le cas de toutes les théories mathématiques). Ceci permet alors de définir un algorithme qui -s'il est poursuivi indé-

finiment- donne une liste de tous les théorèmes : il suffit d'énumérer de façon simple et exhaustive toutes les suites (finies) de formules, de retenir celles qui sont des preuves et d'ajouter les formules qui y figurent à la liste des théorèmes.

De cette façon, l'ensemble des théorèmes apparaît comme effectivement énumérable , nous dirons aussi récursivement énumérable.

La notion de grammaire -introduite par E.POST en 1943- est un des moyens qui permet de traduire cette notion intuitive de langage récursivement énumérable en généralisant la construction de l'ensemble des théorèmes.

DEFINITION.

Une grammaire G est la donnée

(i) - d'un alphabet fini comportant des lettres dites terminales a,b,c,... et des lettres dites variables A,B,C,... ; une des variables, notée S, (pour START), est appelée variable de départ.

(ii) - d'une famille finie de productions de la forme $u \longrightarrow v$, u et v étant des mots de l'alphabet de G, une variable au moins apparaissant dans u.

A une grammaire G nous associons un langage $L(G)$ comme suit :

1) - Un mot β est dit déduit élémentairement du mot α s'il existe un sous-mot u de α (c'est-à-dire un segment de lettres contigües dans α) et une production $u \longrightarrow v$ de G tels que β soit le résultat du remplacement de u par v dans α ; on écrit $\alpha \longrightarrow \beta$.

2) - Un mot β est dit déduit de α s'il existe une suite de mots $\alpha_1, \ldots, \alpha_n$ tels que $\alpha_1 = \alpha$, $\alpha_n = \beta$ et, pour chaque i, α_{i+1} est dé-

duit élémentairement de α_i. On écrit $\alpha \xrightarrow{*} \beta$.

3) - Le langage L(G) est alors l'ensemble des mots déduits du mot S (mot à une seule lettre) et formés de lettres terminales :

$$L(G) = \{\alpha \in \underbrace{\{a,b, \ldots\}^*}_{\text{notation usuelle de l'ensemble des mots écrits avec a,b,c, ...}} : S \longrightarrow \alpha\}$$

1.2. Quelques exemples.

Nous noterons λ le mot vide et écrirons :

$$u \longrightarrow v \mid w \mid x \ldots$$

en place de la famille des productions :

$$u \longrightarrow v, \quad u \longrightarrow w, \quad u \longrightarrow x, \ \ldots$$

Nous noterons aussi $|u|$ la longueur du mot u.

(1) G
- 1 — lettre terminale
- S — variable
- $S \longrightarrow \lambda \mid aaaS$ — productions.

L(G) est l'ensemble des suites de 1 de longueur divisible par 3.

(2) G
- a,b
- S
- $S \longrightarrow bS \mid abS \mid aabS \mid aa \mid a \mid \lambda$

L(G) est l'ensemble des mots de $\{a,b\}^*$ dans lesquels ne figurent jamais trois lettres a consécutives.

(3) G
- 0, 1, (,), v
- S, A
- $S \longrightarrow (v\ 0) \mid (v\ 1\ A$
- $A \longrightarrow 0\ A \mid 1\ A \mid)$

L(G) est l'ensemble des mots (v α) où α est l'écriture en binaire

d'un entier.

Si dans une théorie mathématique on ne veut pas introduire un alphabet infini pour écrire les variables v_o, v_1, ..., l'ensemble L(G) est un codage commode de ces variables avec un alphabet fini.

(4) G
- a,b
- S
- S ⟶ ab|aSb

L(G) est l'ensemble des mots $a^n b^n$, $n \geqslant 1$.

(5) G
- a,b
- S
- S ⟶ λ|Sab|aSb|abS|Sba|bSa|baS

L(G) est l'ensemble des mots contenant autant de lettres a que de lettres b.

(6) G
- (,), [,]
- S
- S ⟶ ( )|[ ]|(S)|[S]|SS

L(G), appelé langage de DYCK, est formé des écritures équilibrées de parenthèses et de crochets ; par exemple, le mot [( )]( ) peut se déduire comme suit :

S ⟶ SS ⟶ [S]S ⟶ [( )]S ⟶ [( )]( )

(7) G
- (,), 0, s, +, ., v, 1
- T, V, A, T est la lettre de départ
- V ⟶ (v 0)|(v 1 A
- A ⟶ 0 A|1 A|)
- T ⟶ 0|V|s(T)|(T) + (T)|(T).(T)

Les mots déduits de V sont des codages des variables v_o, v_1, ... ; le langage L(G) est celui des termes de l'arithmétique.

En ajoutant à G

- les lettres terminales $\wedge, \vee, \neg, \longrightarrow, =, \prec, \preccurlyeq$,
- la variable S prise comme lettre de départ,
- les productions
  - $S \longrightarrow S = S \mid S \prec S \mid S \preccurlyeq S$
  - $S \longrightarrow \neg S \mid S \wedge S \mid S \vee S \mid S \longrightarrow S$,

on obtient une grammaire G' dont le langage L(G') est celui des formules de l'arithmétique.

(8) G
- a,b
- S, T, X, A, B
- $S \longrightarrow T$
- $T \longrightarrow XTX \mid a \mid b$
- $X \longrightarrow a \mid b$
- $S \longrightarrow AB \mid BA$
- $A \longrightarrow a \mid XAX$
- $B \longrightarrow b \mid XBX$

Les trois premières séries de productions donnent les mots de longueur impaire sur l'alphabet a,b.

Les trois autres donnent les mots de la forme :

u a v u' b v' et u b v u' a v' où $|u| = |v|$ et $|u'| = |v'|$

c'est-à-dire aussi :

u a w w' b v' et u b w w' a v' où $|u| = |w'|$ et $|w| = |v'|$

On en déduit que L(G) est l'ensemble des mots qui ne sont pas de la forme uu (écrits sur l'alphabet a,b).

(9) G
- a,b,c
- S,B
- $S \longrightarrow aBc$
- $aB \longrightarrow aaBcB$
- $cB \longrightarrow Bc \quad aB \longrightarrow ab \quad bB \longrightarrow bb$

L(G) est l'ensemble des mots $a^n b^n c^n$, $n \geqslant 1$.

(10) G [a,b

S, T, A, B, D, D', F, F'

[S ⟶ aa|bb|aDT|bD'T|λ

T ⟶ aAT|bBT|Fa|F'b

Xa ⟶ aX et Xb ⟶ bX pour X = A,B,D,D'

AF ⟶ Fa BF ⟶ Fb

DF ⟶ aa D'F' ⟶ bb

DF' ⟶ ba D'F ⟶ ab]

L(G) est l'ensemble des mots d'alphabet a,b qui sont de la forme uu.

(11) G [0, 1, =, +

S, A, R, Z, Z', Z", U, U', U"

S ⟶ + A =

[A ⟶ ZZ'Z"|UZ'U"|ZU'U"

A ⟶ AZZ'Z"|AUZ'U"|AZU'U"|RUU'Z"

R ⟶ AZZ'U"|RUZ'Z"|RZU'Z"|RUU'U"

X'Y ⟶ YX' X"Y ⟶ YX" X"Y' ⟶ Y'X"
(les lettres X et Y étant Z ou U)

+ Z ⟶ 0 + + U ⟶ 1 +

Z' ⟶ 0 U' ⟶ 1

Z" = ⟶ = 0 U" = ⟶ = 1]

L(G) est l'ensemble des écritures

$$a_1\ a_2 \ldots a_n + b_1\ b_2 \ldots b_n = c_1\ c_2 \ldots c_n$$

représentant trois nombres en binaire (avec peut-être certains zéros inutiles en tête) dont les deux premiers ont pour somme le troisième.

1.3. Les exemples précédents montrent que les grammaires permettent de faire des duplications, des transports, etc.... C'est ainsi que -compliquant ce qui a été fait à l'exemple 1.1.- on peut représenter en écriture bi-

naire l'ensemble diophantien

$$\{x \in \mathbb{N} : \exists\, y_1 \in \mathbb{N} \ldots \exists\, y_k \in \mathbb{N} \quad P(x, y_1, \ldots, y_k) = 0\}$$

(P étant un polynôme à coefficients entiers relatifs).

Développant les techniques des exemples 7 et 10, il est également possible d'associer une grammaire au langage formé des théorèmes d'une théorie mathématique.

En fait, les langages définis par les grammaires sont *tous* les langages récursivement énumérables. Ceci peut se montrer soit en utilisant le résultat de MATIASSEVITCH sur la représentation diophantienne des récursivement énumérables, soit en mimant le comportement d'une machine de Turing par une grammaire.

Au vu des exemples 1 à 11 certaines grammaires apparaissent plus simples que d'autres ; les contraintes que vérifient leurs productions traduisent quelques caractères des langages de la pratique mathématique et/ou informatique. Quatre classes de langages sont particulièrement remarquables ; elles définissent la hiérarchie de Noam CHOMSKY (définie par ce dernier vers 1956).

1). Grammaire de type 0.

Aucune contrainte n'est demandée ; ces grammaires donnent la classe de tous les langages récursivement énumérables.

2). Grammaire de type 1 (ou context-sensitive)

Les productions sont toutes du type $uAv \longrightarrow u\alpha v$ où A est une variable et α un mot non vide.

Ces productions opèrent sur une seule variable mais prise dans le contexte u à gauche et v à droite, d'où l'appellation anglaise

"context-sensitive". (La clause "α non vide" est essentielle pour ne pas obtenir tous les langages récursivement énumérables ; elle interdit la dérivation du mot vide, ce que l'on peut circonscrire par un amendement ad hoc : on permet la production $S \longrightarrow \lambda$ à condition que S n'apparaisse dans aucun membre droit de production).

3). Grammaire de type 2 (ou context free).

Les productions sont ici de la forme $A \longrightarrow \alpha$; elles opèrent donc sur une seule lettre, indépendamment du contexte (d'où l'appellation "context-free" en anglais).

4). Grammaire de type 3.

Les productions sont toutes de la forme $A \longrightarrow \alpha B$ où A et B sont des variables (peut être identiques) et α un mot formé de lettres terminales.

Ces grammaires, dites régulières, définissent les langages reconnaissables par automate fini.

§ 2 - LES LANGAGES "CONTEXT-SENSITIVE" (CS)

2.1. Les grammaires CS ont la propriété de monotonie suivante :

(*) pour toute production $u \longrightarrow v$ on a $|u| \leq |v|$

Pour ce qui est des langages définis, les grammaires monotones sont équivalentes aux CS : on peut, par exemple, remplacer une production monotone comme $XY \longrightarrow ZTU$ par les productions CS suivantes :

$$\begin{cases} XY \longrightarrow \hat{X}Y \\ \hat{X}Y \longrightarrow \overset{\wedge\wedge}{X}Y \\ \overset{\wedge\wedge}{X}Y \longrightarrow \hat{Z}Y \\ \hat{Z}Y \longrightarrow ZTU \end{cases}$$

De cette manière, on voit que tous les exemples du § 1 sont des langages CS.

2.2. La propriété de monotonie a la conséquence suivante : si $u \in L(G)$ alors il existe une dérivation $S \longrightarrow u_1 \longrightarrow u_2 \longrightarrow \ldots \longrightarrow u_n \longrightarrow u$ formée de mots distincts de longueurs au plus celle de u.

Ces dernières dérivations sont en nombre fini et facilement énumérables ; ceci permet alors de décider l'appartenance d'un mot quelconque u au langage $L(G)$.

Il est assez intuitif que l'algorithme ainsi défini se transcrit en programme d'une machine de Turing pour laquelle

($\Diamond$) [s'il existe un calcul acceptant un mot u alors il y en a un utilisant un espace au plus égal à la longueur de u.

S.Y.KURODA a prouvé (en 1964) la réciproque :

$$CS = NSPACE(n).$$

i.e. les langages CS sont exactement ceux reconnus en espace linéaire par une machine de Turing non déterministe (cf. l'exposé de J.STERN du même volume sur cette notion).

2.3. Il est assez facile de voir que les langages CS sont stables par réunion, produit ($L.L'$ est par définition l'ensemble des mots obtenus par juxtaposition d'un mot de L et d'un mot de L') et l'opération $*$ de KLEENE (par définition $L^* = \{\lambda\} \cup L \cup L^2 \cup L^3 \cup \ldots$).

La stabilité par intersection se voit plus aisément en raisonnant sur la classe NSPACE(n).

Si la classe DSPACE(n) -classe des langages reconnus en espace linéaire par des machines de Turing *déterministes*- est clairement close par complémentation, on ne sait pas si c'est le cas de la classe NSPACE(n) ;

les deux questions suivantes sont ouvertes :

| co - CS $\stackrel{?}{=}$ CS (i.e. CS est-elle close par complémentation ?)

| DSPACE(n) $\stackrel{?}{=}$ NSPACE(n) (i.e. DSPACE(n) est-elle strictement incluse dans NSPACE(n) ?).

2.4. Appelons homomorphisme toute application h de Σ^* dans Δ^* (Σ^* et Δ^* sont les ensembles des mots écrits dans les alphabets Σ et Δ) qui préserve le produit (la juxtaposition -ou concaténation- de mots) ; une telle application est déterminée par les mots images des lettres de Σ.

On voit facilement que :

1) - Si L est CS alors $h^{-1}(L) = \{u \in \Sigma^* : h(u) \in L\}$ est aussi CS.

2) - Si h est non effaçant -c'est-à-dire si h envoie chaque lettre sur un mot non vide- alors h(L) est CS.

En revanche, tout langage récursivement énumérable est l'image d'un CS par un homomorphisme effaçant : soit G une grammaire quelconque ; considérons un nouveau symbole terminal z et une nouvelle variable Z et posons :

(i) G' est la grammaire dont les productions sont :

- $ZA \longrightarrow AZ$ pour chaque variable de G
- $Z \longrightarrow z$
- $u \longrightarrow v\underbrace{ZZ \ldots Z}_{|u| \text{ fois } Z}$ pour chaque production $u \longrightarrow v$ de G

(ii) h est l'homomorphisme qui efface z et laisse les autres lettres inchangées.

Il est immédiat que L(G) = h(L(G')) ; par ailleurs G' étant monotone, L(G) est CS.

§ 3 - LES LANGAGES REGULIERS.

3.1. Notons d'abord que toute grammaire régulière peut être transformée -sans changer le langage associé- de sorte que toutes ses productions soient de la forme $A \longrightarrow \lambda$, $A \longrightarrow a$ ou $A \longrightarrow aB$:
par exemple une production $A \longrightarrow abcB$ se remplace par les trois productions $A \longrightarrow aA'$, $A' \longrightarrow bA''$, $A'' \longrightarrow cB$.

Cette forme permet de voir facilement la relation avec les automates finis.

Un automate fini $\mathscr{A}$ est une machine à mémoire finie, traduite par un ensemble fini Q d'états, fonctionnant comme suit :
1) - Un état initial et un ou plusieurs états finaux sont distingués dans Q.
2) - pour chaque lettre a de l'alphabet (fini) des mots d'entrée et chaque état q est associé une famille (éventuellement vide) d'états $\delta(q,a)$. Si cette famille n'a jamais plus d'un seul élément, $\mathscr{A}$ est dit déterministe.
3) - $\mathscr{A}$ lit successivement les lettres du mot entré. Partant de l'état initial, $\mathscr{A}$ évolue au fil de ses lectures dans différents états de sorte que la lecture de a dans l'état q l'amène à un état quelconque q' dans $\delta(q,a)$.

(Si $\delta(q,a)$ est vide, l'automate se bloque).

4) - Un mot u est dit accepté par $\mathscr{A}$ s'il existe un calcul de $\mathscr{A}$ sur le mot d'entrée u qui amène $\mathscr{A}$ -à l'issue de la lecture de u- sur un état final.

La traduction d'une grammaire G en automate fini $\mathscr{A}$ est simple :

(i) $\mathscr{A}$ a un état final q et des états q_A indexés par les variables de G,

(ii) q_S est l'état initial de $\mathscr{A}$

(iii) $\delta(q,a)$ est vide et $\delta(q_A, a)$ contient q_B (resp. q) si et seulement si $A \longrightarrow aB$ (resp. $A \longrightarrow a$) est une production de G.

Il est clair que L(G) est le langage accepté par $\mathscr{A}$.

Cette notion d'automate fini a été introduite par S.KLEENE en 1956, développant des idées dégagées par les travaux de Mc CULLOCH et PITTS sur le fonctionnement des cellules nerveuses (en 1943).

L'équivalence -quant aux langages reconnus- des automates finis déterministes et non déterministes a été montré par M.RABIN et D.SCOTT (1959) : l'idée consiste à passer d'un ensemble d'états Q à l'ensemble des parties de Q pour rendre déterministe un automate.

Nous renvoyons à l'article de D.PERRIN et J.SAKAROVITCH dans le même volume pour l'illustration de cette notion d'automate fini.

3.2. Les langages réguliers sont stables par réunion, intersection, produit, opération *, homomorphisme et homomorphisme inverse.

KLEENE a montré que *l'on peut caractériser les langages réguliers comme la plus petite classe de langages contenant la langage vide et les langages réduits à un seul mot d'une seule lettre, et fermée par réunion, produit et opération* *.

Indiquons le principe de la preuve de ce résultat.

1) - Soit G une grammaire régulière ; notons X(A) l'ensemble des mots (de l'alphabet des lettres terminales) déduits de la variable A. On peut traduire les productions de G par des équations ensemblistes :
si $A \longrightarrow b_1 \mid \ldots \mid b_p \mid a_1 B_1 \mid \ldots \mid a_n B_n$ sont toutes les productions de

premier membre la lettre A, tout mot déduit de A l'est via une de ces productions, d'où l'équation ensembliste :

$$X(A) = \{b_1, \ldots, b_p\} \cup a_1 . X(B_1) \cup \ldots \cup a_n . X(B_n)$$

2) - Traduisant ainsi G, on obtient un système d'équations ayant un caractère "linéaire". Ce système se résout par des applications successives du Fait suivant

Fait : Soient E *et* F *deux ensembles de mots ; si* F *ne contient pas le mot vide alors l'équation ensembliste* $X = E \cup F.X$ *a une solution unique* $X = F^*.E$.

Preuve.

On vérifie aisément que $F^*.E$ est solution. Réciproquement, si X est solution alors :

$$\begin{aligned} X &= E \cup F.X \\ &= E \cup F.(E \cup F.X) = E \cup F.E \cup F^2.X \\ &= \ldots \\ &= E \cup F.E \cup F^2.E \cup \ldots \cup F^n.E \cup F^{n+1}.X \\ &= (\{\lambda\} \cup F \cup F^2 \cup \ldots \cup F^n).E \cup F^{n+1}.X \end{aligned}$$

Ceci étant vrai pour tout n, on a donc :

$$X = F^*.E \cup F^p.X \quad \text{(pour tout } p \geqslant 1)$$

et donc :

$$\begin{aligned} X &= \bigcap_{p \geqslant 1} [F^*.E \cup F^p.X] \\ &= F^*.E \cup \left(\left[\bigcap_{p \geqslant 1} F^p\right].X\right) \\ &= F^*.E \end{aligned}$$

car -le mot vide n'étant pas dans F- les mots de F^p sont de longueur au moins p.

Nous verrons plus loin (cf. alinéa 4.4) une autre traduction en termes de systèmes d'équations linéaires dans le semi-anneau des séries

formelles $\mathbb{N}[[\Sigma]]$. C'est cette interprétation algébrique qui a conduit M.P.SCHUTZENBERGER et S.EILENBERG à appeler rationnels les langages réguliers.

3.3. La grande simplicité des langages réguliers impose un résultat de limitation très fort, appelé lemme de la pompe :

THEOREME.

Soit L un langage régulier ; il existe un entier n tel que, pour tout mot u de longueur au moins n, si le mot xuy est dans L alors on peut écrire $u = svt$ de sorte que $xsv^i tz$ est dans L pour tout $i \in \mathbb{N}$, v étant non vide.

Preuve.

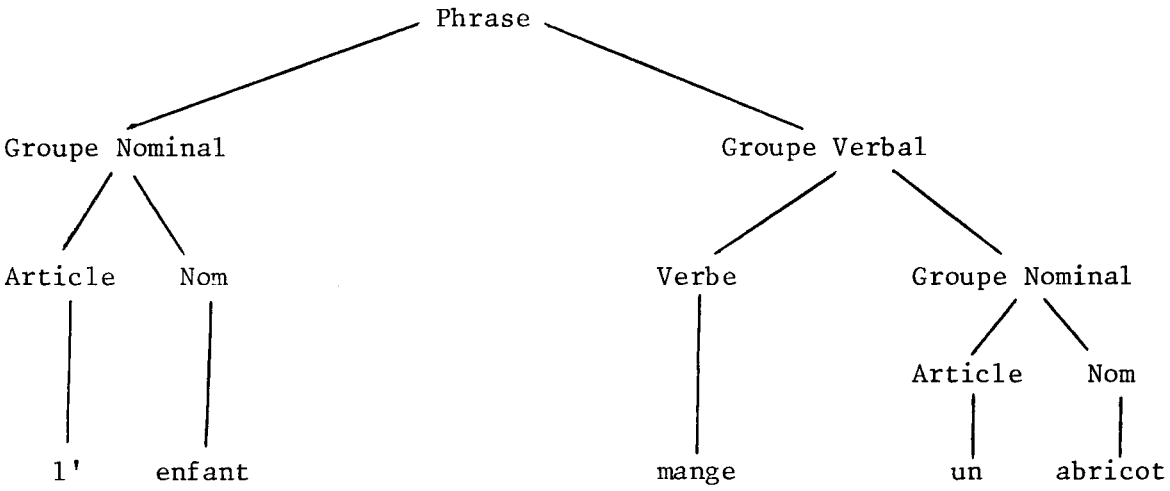
Soit $\mathscr{A}$ un automate fini acceptant L ; on prend pour n un entier supérieur au nombre d'états de $\mathscr{A}$.

Considérons un calcul de $\mathscr{A}$ qui accepte le mot xuy. Si u s'écrit comme la suite des lettres $a_1\ a_2 \ldots a_\ell$, avec $\ell \geqslant n$ et donc ℓ supérieur au nombre d'états de $\mathscr{A}$, il existe deux indices j et k tels que $j < k$ et les lectures des mots $x\ a_1 \ldots a_j$ et $x\ a_1 \ldots a_j \ldots a_k$ amènent -via le calcul retenu de l'automate- dans le même état q.

La suppression ou la multiplication des lectures de $a_{j+1} \ldots a_k$ ne change pas la transition d'états ; ainsi, posant $s = a_1 \ldots a_j$, $v = a_{j+1} \ldots a_k$ et $t = a_{k+1} \ldots a_\ell$, on a $u = svt$, $v \neq \lambda$ et, pour tout $i \in \mathbb{N}$, $xsv^i ty \in L$.

§ 4 - LES LANGAGES "CONTEXT FREE (CF).

4.1. C'est dans les travaux de linguistes tels BLOCH , HARRIS , WELLS (durant les années 40-50) que se trouve l'origine de la théorie des langages CF. L'exemple ci-dessous est typique de l'analyse syntaxique d'une phrase d'une langue naturelle mettant en évidence une grammaire CF.



Mise en place par N.CHOMSKY vers 1956, cette classe des CF est rencontrée -sous d'autres versions- dans l'analyse syntaxique des langages de programmation (BACKUS, NAUR, 1960) et les problèmes de traduction automatique (OETTINGER, 1960). L'équivalence des différentes présentations et l'association de ces langages avec les automates à pile (CHOMSKY, SCHUTZENBERGER, EVEY) ne sont achevées que vers 1963.

4.2. Si G est une grammaire CF, à toute dérivation $S \to u_1 \to \ldots \to u_n \to u$ d'un mot u on peut associer un arbre (cf. l'exemple de 4.1 et les exemples ci-dessous) ; cette association n'est pas injective : elle permet d'oublier l'ordre relatif de certaines dérivations élémentaires et de ne retenir que l'essentiel du mode de construction du mot.

Exemples.

G [a,b

S, A, T, U

[$S \longrightarrow AT \mid UA$

$T \longrightarrow ba \mid bTa$

$U \longrightarrow ab \mid aUb$

$A \longrightarrow a \mid aA$

On voit facilement que :

$L(G) = \{a^n b^p a^q : n,p,q \geqslant 1 \quad \text{et} \quad (n = p \quad \text{ou} \quad p = q)\}$

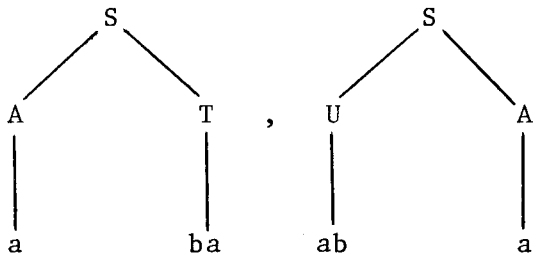
Consdérons les trois dérivations suivantes du mot aba :

$S \longrightarrow AT \longrightarrow aT \longrightarrow aba$

$S \longrightarrow AT \longrightarrow Aba \longrightarrow aba$

$S \longrightarrow UA \longrightarrow abA \longrightarrow aba$;

les deux premières correspondent à un même arbre, mais pas la troisième :



Lorsque tout mot de $L(G)$ n'a qu'un seul arbre de dérivation, la grammaire G est dite non ambigüe ; dans le cas contraire, le nombre d'arbres de dérivation d'un mot est appelé son coefficient d'ambiguïté relativement à G.

Si G ne contient aucune production de type $A \longrightarrow B$ ou $A \longrightarrow \lambda$ (ce à quoi on peut toujours se ramener si $L(G)$ ne contient pas le mot vide, ou bien en acceptant la production $S \longrightarrow \lambda$ et en interdisant toute apparition de S à droite d'une production) alors le degré d'ambiguï-

té d'un mot de L(G) est toujours fini.

L'ambiguïté est une notion attachée aux grammaires et non aux langages : un même langage peut être construit par une grammaire ambigüe et une autre non ambigüe. On peut cependant montrer que certains langages -c'est le cas de l'exemple ci-dessus- sont ambigüs inhérents : toute grammaire CF les construisant est ambigüe.

4.3. On peut traduire les productions d'une grammaire CF en termes d'équations comme à l'alinéa 3.2..

Considérons l'exemple suivant : G $\begin{cases} a,b \\ S,B \\ S \longrightarrow a \mid aB \mid BS \\ B \longrightarrow b \mid SB \end{cases}$

Si l'on note X(S) et X(B) les ensembles de mots de l'alphabet terminal déduits de S et B (bien sûr, X(S) est L(G)) alors X(S) et X(B) sont solutions du système d'équations ensemblistes :

$$\begin{cases} X = \{a\} \cup aY \cup YX \\ Y = \{b\} \cup XY \end{cases}$$

Ce système n'a pas une unique solution ($X = Y = \{a,b\}^*$ est aussi solution) mais (X(S), X(B)) en *est la plus petite solution* : ces équations impliquant en effet $X \supset X(S)$ et $Y \supset X(B)$.

4.4. Une traduction plus intéressante (développée par SCHUTZENBERGER) est en termes de séries formelles de $\mathbb{N}[[\mathcal{T}]]$, $\mathcal{T}$ étant l'alphabet terminal de la grammaire.

Considérons l'exemple suivant : G $\begin{cases} a,b \\ S,B \\ S \longrightarrow aB \mid Ba \\ B \longrightarrow aBB \mid BaB \mid BBa \mid b \end{cases}$

Aux variables S et B associons les séries formelles σ_S et σ_B :

$$\sigma_S = \sum_{u \in \mathcal{T}^*} \gamma_u u \ , \quad \sigma_B = \sum_{u \in \mathcal{T}^*} \delta_u u$$

γ_u et δ_u étant les nombres des arbres de dérivation du mot u à partir de S et B (ces coefficients valent 0 si u n'est pas dérivable à partir de S ou B).

On peut montrer que ces séries formelles σ_S et σ_B sont solutions du système d'équations algébriques dans $\mathbb{N}[[\mathcal{T}]]$:

$$\begin{cases} X = aY + Ya \\ Y = aYY + YaY + YYa + b \ ; \end{cases}$$

de plus, ces solutions sont les seules dont les coefficients du mot vide soient nuls.

4.5. Sans changer les langages associés, on peut mettre les grammaires CF sous des formes normales très simples :

1). Forme normale de CHOMSKY :

toutes les productions sont du type $A \longrightarrow \lambda$, $A \longrightarrow a$ ou $A \longrightarrow BC$.

2). Forme normale de GREIBACH :

toutes les productions sont de type $A \longrightarrow a$, $A \longrightarrow aB$ ou $A \longrightarrow aBC$ (avec éventuellement $S \longrightarrow \lambda$ si S n'apparaît nulle part à droite).

La preuve de la forme normale de CHOMSKY est facile : par exemple, une production $A \longrightarrow aBAC$ peut se remplacer par :

$$A \longrightarrow A'C$$

$$A' \longrightarrow A''A$$

$$A'' \longrightarrow A'''B$$

$$A''' \longrightarrow a$$

La preuve de la forme normale de GREIBACH est plus délicate ; nous esquissons une preuve possible sur un exemple.

Soit G la grammaire $\begin{cases} a,b \\ S,B \\ \begin{cases} S \longrightarrow a \mid BS \\ B \longrightarrow b \mid SB \end{cases} \end{cases}$;

reprenant l'interprétation ensembliste indiquée en 4.3., nous noterons X(S) et X(B) les ensembles des mots de $\{a,b\}^*$ déduits de S et B. Ces ensembles sont solutions (minimales) du système d'équations ensemblistes :

$$(1) \quad \begin{cases} X = \{a\} \cup YX \\ Y = \{b\} \cup XY \end{cases}$$

Avec la convention de considérer union et concaténation d'ensembles de mots comme addition et multiplication, on peut écrire ce système sous forme matricielle :

$$(X\ Y) = (\{a\}\ \{b\}) + (X\ Y) \begin{pmatrix} \emptyset & Y \\ X & \emptyset \end{pmatrix}$$

$$= K + (X\ Y)\ M$$

$$= K + [K + (X\ Y)\ M]\ M$$

$$= K + KM + (X\ Y)\ M^2$$

$$= .....$$

$$= K + K(M + M^2 + ... + M^n) + (X\ Y)\ M^{n+1}$$

Comme les coefficients de M^{n+1} sont des ensembles vides ou formés de mots de longueur au moins n, on déduit (comme en 3.2. pour le Fait)

$$(X\ Y) = K + K(M + M^2 + ... + M^n + ...)$$

Appelons Z,T,U,V les ensembles -dépendant de X et Y- qui sont les coefficients de $H = M + M^2 + ...$.

On a $H = M + MH$, les ensembles X,Y,Z,T,U,V sont donc solutions

du système

$$(2)\quad \begin{cases} (X\ Y) = (\{a\}\ \{b\}) + (\{a\}\ \{b\}) \begin{pmatrix} Z & U \\ T & V \end{pmatrix} \\ \begin{pmatrix} Z & U \\ T & V \end{pmatrix} = \begin{pmatrix} \emptyset & Y \\ X & \emptyset \end{pmatrix} + \begin{pmatrix} \emptyset & Y \\ X & \emptyset \end{pmatrix} \begin{pmatrix} Z & U \\ T & V \end{pmatrix} \end{cases}$$

soit aussi

$$(3)\quad \begin{cases} X = \{a\} \cup \{a\}\ Z \cup \{b\}\ T \\ Y = \{b\} \cup \{a\}\ U \cup \{b\}\ V \\ Z = YT \\ T = X \cup XZ \\ U = Y \cup YV \\ V = XU \end{cases}$$

ou

$$(4)\quad \begin{cases} X = \{a\} \cup \{a\}\ Z \cup \{b\}\ T \\ Y = \{b\} \cup \{a\}\ U \cup \{b\}\ V \\ Z = \{b\}\ T \cup \{a\}\ UT \cup \{b\}\ VT \\ T = \{a\} \cup \{a\}\ Z \cup \{b\}\ T \cup \{a\}\ ZZ \cup \{b\}\ TZ \\ U = \{b\} \cup \{a\}\ U \cup \{b\}\ V \cup \{a\}\ UV \cup \{b\}\ VV \\ V = \{a\} \cup \{a\}\ ZU \cup \{b\}\ TU \end{cases}$$

Les systèmes (1) et (4) ont les mêmes solutions en X et Y, donc les mêmes solutions minimales ; il en résulte que $L(G) = L(G')$ où G' est la grammaire sous forme de GREIBACH :

$$G' \begin{cases} a,b \\ S, B, C, D, E, F \\ \begin{cases} S \longrightarrow a \mid aC \mid bD \\ B \longrightarrow b \mid aE \mid bF \\ C \longrightarrow bD \mid aED \mid bFD \\ D \longrightarrow a \mid aC \mid bD \mid aCC \mid bDC \\ E \longrightarrow b \mid aE \mid bF \mid aEF \mid bFF \\ F \longrightarrow aE \mid aCE \mid bDE \end{cases} \end{cases}$$

On remarque que la forme de GREIBACH est au plus proche de celle des grammaires régulières ; elle constitue un outil très puissant dans l'étude des langages CF.

4.6. La classe des langages CF est close par réunion, produit, opération $*$, homomorphisme et homomorphisme inverse.

Indiquons quelques preuves via des grammaires :

1) - On a $L(G)^* = L(G')$ où

(i) G' a une lettre de variable de plus que G, cette lettre -soit T- est la lettre de départ de G'.

(ii) Les productions de G' sont celles de G ainsi que $T \longrightarrow S \mid TT$.

2) - Si h est un homomorphisme on a $h(L(G)) = L(G')$ où

(i) G' a pour variables celles de G ainsi que des variables A_a, A_b, ... associées aux lettres terminales de G.

(ii) Les productions de G' sont celles de G dans lesquelles les variables A_a, A_b, ... se substituent aux lettres terminales a, b, ..., ainsi que les productions $A_a \longrightarrow h(a)$, $A_b \longrightarrow h(b)$, ...

3) - La clôture par homomorphisme inverse est moins facile. Supposons G sous forme normale de GREIBACH et soient $u_1, \ldots, u_n$ les images par h des lettres $a_1, \ldots, a_n$. Considérons toutes les dérivations dans G de la forme $A \xrightarrow{*} u_i \; B_1 \ldots B_k$ (avec nécessairement k au plus le double de la longueur de u_i) ; en remplaçant u_i par a_i on obtient une grammaire G' telle que $L(G') = h^{-1}(L(G))$.

L'intersection de deux langages CF n'est pas en général CF : par exemple, les deux langages $\{a^n b^n a^p : n, p \geqslant 1\}$ et $\{a^n b^p a^p : n, p \geqslant 1\}$ sont CF mais leur intersection qui est $\{a^n b^n c^n : n \geqslant 1\}$ ne l'est

pas (cf. l'alinéa 4.7.). Il est en fait possible de montrer que tout langage récursivement énumérable est image homomorphe de l'intersection de deux CF.

En revanche, l'intersection d'un langage CF avec un langage régulier est CF ; on peut voir ce résultat important comme suit. Soit K un langage régulier reconnu par l'automate fini $\mathscr{A}$ et soit G une grammaire sous forme de GREIBACH ; considérons la grammaire G' suivante :

(i) Les variables X_{Aq} de G' sont indexées par les variables de A et les états de $\mathscr{A}$; il y a, en plus, une variable de départ I.

(ii) Les lettres terminales de G' sont celles de G, les lettres q_o, q_1 ,..., q_n désignant les états de $\mathscr{A}$ et le symbole $=$.

(iii) Les productions de G' sont :

$I \longrightarrow = X_{S\, q_o} =$	q_o étant l'état initial de $\mathscr{A}$
$X_{Aq} \longrightarrow a\, X_{Bq'}$	si $A \longrightarrow aB$ est une production de G et $q' \in \delta(q,a)$ (δ est la fonction de transition de $\mathscr{A}$)
$X_{Aq} \longrightarrow a\, X_{Bq'}\, X_{Cq''}$	si $A \longrightarrow aBC$ est une production de G et $q' \in \delta(q,a)$, q'' quelconque
$X_{Aq} \longrightarrow qaq'$	si $A \longrightarrow a$ est dans G et $q' \in \delta(q,a)$.

Il est facile de voir que $L(G) \cap K = \theta(\varphi^{-1}(L(G'))$ où l'homomorphisme θ efface les symboles autres que les terminaux de G (et laisse inchangés ces derniers) et où φ envoie la lettre q_i sur le mot $q_i\, q_i$ et introduit les mots $= q_o$ et $q =$ pour q état final de $\mathscr{A}$.

Le complémentaire d'un langage CF n'est pas en général CF (sans cela la clôture par réunion montrerait celle par intersection) : par exemple, l'ensemble des mots du type uu n'est pas CF alors que son complé-

mentaire l'est (cf. les exemples 8 et 10 de l'alinéa 1.2.).

4.7. Une généralisation du lemme de la pompe relatif aux langages réguliers est valable pour les CF ; nous en énonçons une version simple.

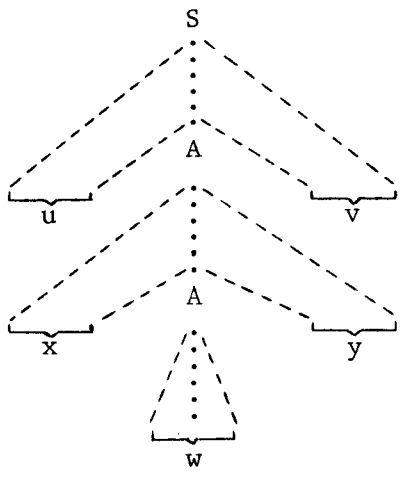
THEOREME.

Soit L un langage CF ; il existe un entier n tel que pour tout mot α de L dont la longueur est au moins n on peut écrire : $\alpha = uxwyv$ où xy n'est pas le mot vide et xwy est de longueur au plus n, et $ux^i wy^i v$ est dans L pour tout $i \in \mathbb{N}$.

Preuve.

Soit G une grammaire sous forme normale de CHOMSKY construisant L ; on prend pour n un entier supérieur à 2^{p+1} où p est le nombre des variables de G.

Considérons l'arbre d'une dérivation de α dans G, c'est un arbre binaire (car G est sous forme de CHOMSKY et les seconds membres des productions ont au plus deux lettres) ; comme la longueur de α est au moins n et donc supérieure à 2^{p+1}, il existe une branche de cet arbre de longueur au moins p+1 ; sur une telle branche il y a au moins une variable A ayant deux occurrences. On a donc :



$S \xrightarrow{*} u\,A\,v$

$A \xrightarrow{*} x\,A\,y$ x y non vide

$A \longrightarrow w$

u,v,w,x,y mots terminaux

$\alpha = uxwyv$ xy non vide

On voit alors facilement que l'on peut dériver de S les mots $u x^i w y^i v$, $i \in \mathbb{N}$. La condition $|xwy| \leq n$ s'obtient si les deux occurrences de A ont été prises le plus bas possible sur la branche la plus longue.

Ce théorème permet facilement de voir que le langage $\{a^n b^n a^n : n \geq 1\}$ n'est pas CF, de même que le langage des mots de la forme uu.

4.8. Appelons D_n le langage -dit de DYCK- des expressions avec n types de parenthèses bien équilibrées ; une grammaire construisant D_n est la suivante

$$\begin{cases} (_1\ ,\ )_1,\ (_2\ ,\ )_2\ ,\ldots,\ (_n\ ,\ )_n \\ S \\ S \longrightarrow SS \mid (_i\ )_i \mid (_i\ S\ )_i \qquad i = 1,2,\ldots,n \end{cases}$$

Le résultat suivant, dû à CHOMSKY et SCHUTZENBERGER, montre que les langages de DYCK sont les CF essentiels.

THEOREME.

Soit L un langage CF ; il existe un entier n, un langage régulier R et un homomorphisme alphabétique h (h envoie une lettre sur une lettre ou bien l'efface) tels que $L = h(D_n \cap R)$.

Nous esquissons la preuve du théorème sur un exemple.

Soit $L = \{a^n b^n ; n \geq 1\}$; une grammaire pour L est

$$\begin{cases} a,b \\ S \\ S \longrightarrow ab \mid aSb \end{cases}$$

ou, sous forme normale de CHOMSKY,

$$
\left[\begin{array}{l}
a,b \\
S,\ A,\ B,\ T \\
\left[\begin{array}{ll}
S \longrightarrow AB & (1) \\
S \longrightarrow AT & (2) \\
T \longrightarrow SB & (3) \\
A \longrightarrow a & (4) \\
B \longrightarrow b & (5)
\end{array}\right]
\end{array}\right]
$$

Nous coderons L dans D_8 dont les parenthèses seront écrites comme suit :

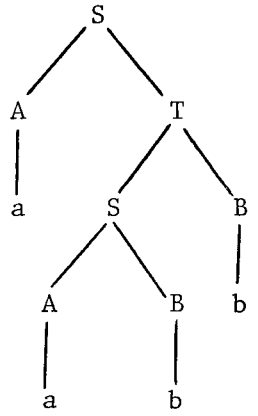
$(_i\ ,\ )_i \quad i=1,2,3,4,5$

$[_i\ ,\ ]_i \quad i=1,2,3$

(les parenthèses sont associées aux cinq productions, les crochets aux productions sans lettre terminale).

A chaque arbre de dérivation d'un mot de L nous associons un mot de D_8 comme l'illustre l'exemple suivant :

considérons l'arbre



de dérivation du mot a^2b^2 ; le codage de cet arbre dans D_8 se fait par étapes traduisant les niveaux de l'arbre :

$(_2\ A\ [_2\ ]_2\ T\ )_2$ application de la production (2)

$(_2\ (_4\ )_4\ [_2\ ]_2\ (_3\ S\ [_3\ ]_3\ B\ )_3\ )_2$ productions (4) et (3)

$(_2\ (_4\ )_4\ [_2\ ]_2\ (_3\ (_1\ A\ [_1\ ]_1\ B\ )_1\ [_3\ ]_3\ (_5\ )_5\ )_3\ )_2$ productions (1) et (5)

$(_2\ (_4\ )_4\ [_2\ ]_2\ (_3\ (_1\ (_4\ )_4\ [_1\ ]_1\ (_5\ )_5\ )_1\ [_3\ ]_3\ (_5\ )_5\ )_3\ )_2$ productions (4) et (5)

ce dernier mot est le code de l'arbre dans D_8.

Il est assez simple de voir que l'ensemble des codes des arbres des dérivations de mots de L est de la forme $D_8 \cap R$ où R est un ensemble régulier ; le langage L lui-même est alors égal à $h(D_8 \cap R)$ où h est l'homomorphisme transformant les lettres $(_4$ et $(_5$ en a et b et effaçant toutes les autres.

Soit φ l'homomorphisme qui envoie les lettres $(_i$ et $)_i$ (i=1,2,...,n) sur les mots :

$$[\underbrace{(\,(\,\ldots\,(\,[}_{i\text{ fois}} \quad \text{et} \quad]\underbrace{)\,)\,\ldots\,)}_{i\text{ fois}}\,] \ ;$$

il est clair que $D_n = \varphi^{-1}(D_2)$. Le théorème précédent assure alors que *tout langage CF est de la forme* $h(\varphi^{-1}(D_2) \cap R)$ *où R est régulier*. Le langage D_2 apparaît ainsi comme le CF essentiel.

S.GREIBACH a prouvé (1973) qu'il existe un langage CF le plus dur possible au sens suivant : tout autre langage CF en est une image inverse par un homomorphisme (au mot vide près).

4.9. Les langages CF peuvent être reconnus par des machines de Turing déterministes en temps polynomial. Nous décrivons ci-dessous, sur un exemple, l'algorithme de COCKE, KASAMI et YOUNGER qui décide si un mot de longueur n est dans un langage L en un nombre d'étapes en $O(n^3)$.

Nous considérons la grammaire suivante (sous forme de CHOMSKY)

G
- a,b
- S, A, B, C
- S ⟶ AB | BC
- A ⟶ a | BA
- B ⟶ b | CC
- c ⟶ a | AB

et le mot u = baaba ; pour déterminer si u est ou non dans L(G), on calcule les ensembles de variables permettant de dériver les différents sous-mots de u, ce qui donne la construction du tableau suivant :

b	a	a	b	a	Sous-mots de longueur 1,
{B}	{A,C}	{A,C}	{B}	{A,C}	
ba	aa	ab	ba		Sous-mots de longueur 2,
{A,S}	{B}	{S,C}	{A,S}		
baa	aab	aba			Sous-mots de longueur 3,
∅	{B}	{B}			
baab	aaba				Sous-mots de longueur 4
∅	{S,A,C}				
baaba					le mot u
{S,A,C}					

cet ensemble contient S, donc baaba est dans L(G).

Le nombre d'étapes de cet algorithme se majore comme suit :

(i) il y a $\frac{1}{2}$ n (n+1) sous-mots de u à considérer (si n = |u|).

(ii) pour chaque sous-mot v de u il faut tester tous les découpages

possibles xy de v, ces découpages sont en nombre plus petit que n.

(iii) en tout il y a donc moins de $\frac{1}{2} n^2 (n+1)$ tests à faire, d'où un temps d'effectuation en $O(n^3)$.

Utilisant l'algorithme rapide de STRASSEN pour la multiplication des matrices, VALIANT a modifié cet algorithme de sorte à avoir un temps de calcul en $O(n^{2,81})$.

On peut aussi montrer que la décision de l'appartenance à un langage CF peut être faite en espace $(\log n)^2$.

4.10. Un type particulier de machines peut être attaché aux langages CF : les automates à pile (en anglais "pushdown automata").

Un tel automate se présente comme suit :

1). Un ruban d'entrée où est écrit un mot dont les lettres sont lues successivement sans retour en arrière.

2). Une mémoire bornée, traduite par un ensemble fini d'états.

3). Une pile où sont mis des lettres, dont à chaque étape l'automate ne peut connaître que la dernière qui y a été mise.

4). Une fonction de transition qui, à

(i) l'état de la machine

(ii) le symbole lu sur le ruban d'entrée

(iii) le symbole figurant en haut de la pile (le dernier rentré), associe un ensemble de possibilités relatives aux

(i) passage à un autre état

(ii) modification de la pile : maintien ou suppression de l'élément du haut de la pile et/ou adjonction de nouveaux symboles en haut de la pile.

Introduits par CHOMSKY et SCHUTZENBERGER en 1963, ces automates

reconnaissent (relativement à diverses formes d'acceptation : par état final, pile vide ou les deux) exactement les langages CF.

Ces automates étant non déterministes, se pose la question de savoir quels sont les langages reconnus par les automates à pile déterministes (c'est-à-dire ceux pour lesquels la fonction de transition donne une *seule* possibilité). On peut démontrer que cette classe des langages "déterministe context-free" (DCF) est strictement incluse dans la classe des CF et est stable par complémentation.

Le problème de déterminer des grammaires rendant compte de ces langages a été résolu par D.KNUTH avec la notion de grammaire LR(k).

Signalons pour conclure un important problème ouvert sur les DCF : y a-t-il un algorithme permettant de décider si deux automates à piles déterministes reconnaissent le même langage ?

La même question a une réponse positive pour les automates finis (ou grammaires régulières) et une réponse négative pour les automates à pile non déterministes (ou grammaires CF).

BIBLIOGRAPHIE.

BERSTEL J.
Transductions and Context-Free Languages,
1979, Teubner.

HARRISON M.
Introduction to Formal Language Theory,
1978, Addison-Wesley.

HOCROFT J. et ULLMAN J.
Introduction to Automata Theory, Languages and Computation,
1979, Addison-Wesley.