Introduction aux grammaires formelles

P. Égré (CNRS, IJN)

Rentrée CogMaster- Septembre 2006

1 Introduction

• Langages naturels et langages formels
• Le programme chomskyen et la grammaire générative (Humboldt 1836: "le langage fait un usage infini de moyens finis").
• Notion de grammaire universelle: "Les principes qui déterminent la forme de la grammaire et qui choisissent une grammaire de forme appropriée sur la base de certains faits constituent un sujet qui pourrait, selon l'usage traditionnel, être appelé "grammaire universelle". L'étude de la grammaire universelle ainsi comprise est une étude de la nature des capacités intellectuelles humaines. Elle tente de formuler les conditions nécessaires et suffisantes qu'un système doit remplir pour être considéré comme une langue humaine potentielle, conditions qui ne sont pas vraies des langues humaines par accident, mais qui sont plutôt enracinées dans la "capacité de langage" de l'homme et qui constituent ainsi l'organisation innée déterminant ce qui compte comme expérience linguistique et ce que la connaissance de la langue élève sur la base de cette expérience" (Chomsky 1967, p. 47-48).
• Éléments de syntaxe formelle, calculabilité

2 Définitions

2.1 Langages

• Un langage consiste généralement en:
  - un *vocabulaire* ou *lexique* A: ensemble de *mots* ou *symboles*
  - des *phrases*: suites finies de mots de A
• Un langage est un ensemble de phrases définies à partir d'un vocabulaire possible.
• Une opération importante est la *concaténation* de mots: $a^\frown b = ab$
• *Mot vide*: e, tel que $a^\frown e = e^\frown a = a$
• Remarques:
a) couramment, les théoriciens des langages formels parlent de symboles là où nous parlons de mots, et de mots là où nous parlons de phrases.

b) une phrase est simplement une suite de mots, on ne dit rien à ce stade de son caractère grammatical ou pas, structuré ou pas.

2.2 Exemples

1) $A = \{a, b\}$
- Des phrases: a, $ababba$, $abbba$,...
- Quelques langages:

$L_1 = \{aa, aba, aaa\}$
$L_2 = \{a^m b^n; m, n \in \mathbb{N}\}$
$L_3 = \{a^n b^n; n \in \mathbb{N}\}$

2) $A = \{\mathsf{un}, \mathsf{une}, \mathsf{homme}, \mathsf{femme}, \mathsf{aime}\}$
- Quelques phrases: $\mathsf{un}^\frown\mathsf{une}$
$\mathsf{femme}^\frown\mathsf{homme}^\frown\mathsf{aime}$
$\mathsf{un}^\frown\mathsf{homme}^\frown\mathsf{aime}^\frown\mathsf{une}^\frown\mathsf{femme}$

2.3 Arbres

Un arbre: une structure $\langle T, \leq \rangle$, avec T un ensemble de nœuds, et $\leq$ une relation réflexive transitive (dite de dominance), telle que :

(i) il y a un nœud qui domine tous les autres (la racine)

(ii) tout élément (autre que la racine) est dominé strictement et immédiatement par un unique nœud [ie a une unique mère]

(iii) la relation $\leq$ est acyclique

Remarque: la condition (iii) implique l'unicité de la racine en (i).

• Par définition, x *domine strictement et immédiatement* y ssi x et y sont distincts, x domine y, et il n'y a pas de nœud z distinct de x et y tel que x domine z et z domine y.

• On peut également ajouter une relation transitive et irréflexive de précédence $\lhd$ sur un arbre (de gauche à droite), ie voir un arbre comme une structure $\langle T, \leq, \lhd \rangle$ telle que:

(iv) x précède y ou inversement ssi x ne domine pas y et y ne domine pas x (exclusion de $\lhd$ et de $\leq$)

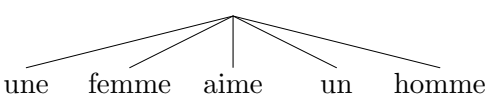
(v) si x précède y, tous les nœuds dominés par x précèdent tous les nœuds dominés par y. (pas de croisement des branches)

Arbre étiqueté: en syntaxe formelle, on s'intéresse à des arbres dont les nœuds peuvent comporter des mots (étiquettes). On peut voir un arbre étiqueté comme une structure $\langle T, \leq , \lhd, Q, L \rangle$ où Q est un ensemble d'étiquettes et L une fonction (éventuellement partielle) qui associe à un nœud une étiquette.

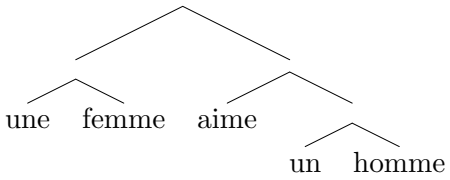
Définitions courantes: x soeur de y; branche; feuille; x mère de y; nœud branchant; nœud terminal (feuille); arbre binaire, ternaire,...

2.4 Exemples

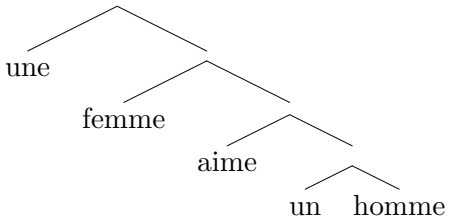
(1)



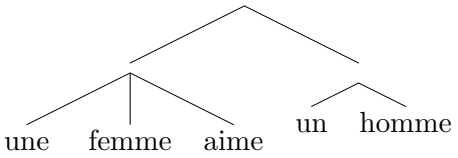
(2)



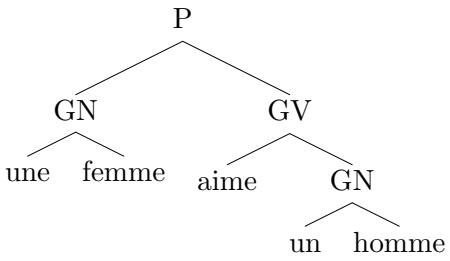
(3)



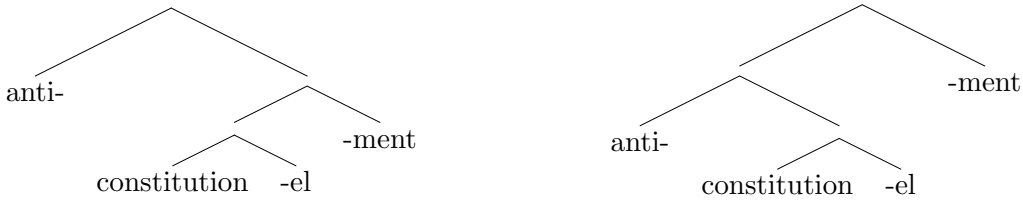
(4)



(5)



(6)



2.5 Présentation linéaire d'un arbre

On manque parfois de place pour écrire un arbre. Dans ce cas, on peut le représenter avec des crochets. Par exemple, on représenterait l'arbre 5 par:

(7) $[_{\text{P}}\ [_{\text{GN}}$ une femme] $[_{\text{GV}}$ aime $[_{\text{GN}}$ un homme]]]

2.6 Exercices

Exercice 1 Traduire les conditions (i)-(v) en logique du premier-ordre avec égalité (en utilisant les symboles $\leq$ et $\lhd$):

• Pour (ii), on représentera par $x < y$ la relation de dominance stricte et immédiate:
- Montrer d'abord qu'on peut définir "x domine strictement et immédiatement y" ($x < y$) à partir de $x \leq y$

- pour l'unicité: traduire "si x est dominé strictement et immédiatement par y et est dominé immédiatement par z, alors y et z sont identiques".

• Pour (iii): traduire "deux nœuds distincts ne peuvent pas se dominer l'un l'autre".

Exercice 2: Relation de *c*-commande. x c-commande y ssi x est soeur d'un nœud qui domine y (ie un nœud c-commande sa soeur et les descendants de sa soeur). Vrai ou faux: Dans l'arbre (2) ci-dessus:

- une c-commande femme
- une c-commande aime
- une c-commande homme
- aime c-commande homme
- aime c-commande un
- homme c-commande un

Exercice 3:
1) Représenter avec des crochets les arbres vus ci-dessus.
2) Donner les arbres correspondant à:
[[un chien] [court [très vite]]]
[[tous [les hommes]] [regardent [la télévision]]]
3) Proposer des étiquettes pour les nœuds branchants de ces deux arbres

3 Grammaire

• Une grammaire est un système qui permet d'engendrer un ensemble de phrases à partir d'un lexique donné. De façon formelle, une grammaire consiste en:
- un lexique *propre* A (symboles terminaux)
- un lexique *intermédiaire* I, comprenant un symbole initial S (symboles de transition)
- un ensemble R de règles de réécriture

• Dérivation d'une phrase dans une grammaire

• Langage engendré par une grammaire: ensemble de toutes les phrases (suites de mots) de A dérivables dans la grammaire à partir de S au moyen des règles de R.

3.1 Exemples

(8) $A = \{a, b\}, I = \{S\}$

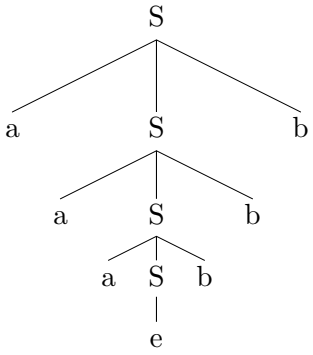
$$\boxed{\begin{array}{l} S \Rightarrow e \\ S \Rightarrow aSb \end{array}} \qquad S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

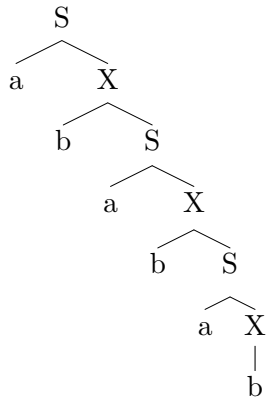
(9) $A = \{a, b\}, I = \{S, X\}$

$$\boxed{\begin{array}{l} S \Rightarrow aX \\ X \Rightarrow b \\ X \Rightarrow bS \end{array}} \qquad S \Rightarrow aX \Rightarrow abS \Rightarrow abaX \Rightarrow ababS \Rightarrow ababaX \Rightarrow abababb$$

3.2 Lien avec les arbres

• *Produit* d'un arbre: la suite formée des feuilles de l'arbre ordonnées par la relation de précédence.





• **Notion de récursivité**: les deux grammaires ci-dessus sont *récursives*, au sens où les règles amènent à répliquer une structure au sein de la structure de départ. Ce processus explique qu'un ensemble fini de règles, sur un vocabulaire fini, puisse engendrer un ensemble infini de phrases.

3.3 Différents types de grammaires

Problème: comment classer les grammaires? Sont-elles toutes aussi expressives? La réponse est négative. Si l'on compare les deux grammaires ci-dessus, on peut voir que:

• dans la seconde grammaire ci-dessus, toutes les règles sont de la forme $A \Rightarrow xB$ ou $A \Rightarrow x$, où A et B sont des symboles intermédiaires et x un symbole terminal (grammaire dite linéaire à droite, cf. l'arbre ci-dessus).

• dans la première grammaire, la règle $S \Rightarrow aSb$ n'est pas de cette forme.

On peut montrer que cette règle n'est pas exprimable sous la forme d'une règle du type $A \Rightarrow xB$ ou $A \Rightarrow x$. Plus généralement, on a ici affaire à deux types différents de grammaires au sein d'une hiérarchie plus vaste (dite de Chomsky):

Grammaire	**Règles**
grammaires linéaires (type 3)	$A \Rightarrow xB$ ou $A \Rightarrow x$
grammaires non-contextuelle (*context-free grammars*, type 2)	$A \Rightarrow \phi$
grammaires contextuelles (*context-sensitive*, Type 1)	$\alpha A \beta \Rightarrow \alpha \phi \beta$ (avec $\phi \neq e$)
grammaires de type 0	$\phi \Rightarrow \psi$

3.4 Exercices

1) Soit $A = \{p, ', \neg, \wedge,), (\}$. Proposer un système de réécriture qui engendre exactement toutes les formules du calcul propositionnel. Utiliser $I = \{S, X\}$. Le $'$ sert pour former les atomes: $p, p', p'', ...$(Indice: X servira a traiter la construction des atomes, et S les formules quelconques).

2) Proposer un système de réécriture de type 3 pour chacun des langages suivants (avec $A = \{a, b\}$):

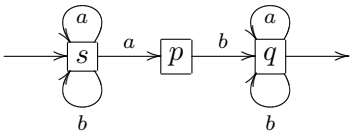
a) $L_1 = \{aa, ab, ba, bb\}$

b) $L_2 = \{x; x$ contient un nombre arbitraire (non-nul) d'occurrences de a et de b, dans n'importe quel ordre$\}$

4 Grammaires linéaires et automates finis

Une grammaire peut être vue comme un système de réécriture, ou bien comme un automate: un mécanisme qui, partant d'un état initial, engendre pas à pas les phrases du langage. La meilleure façon de voir ce qu'est un automate est de commencer par un exemple:

4.1 Un exemple



4.2 Automates finis

Formellement: Un automate fini est un quintuplet $\langle Q, A, E, s, T\rangle$, où Q est un ensemble fini d'états; A est un ensemble fini de symboles ou mots (dans lequel on inclut e par défaut); s désigne l'état initial et T un ensemble d'états terminaux; E est une relation de transition qui associe à un état et un symbole un autre état.

Exemple: ci-dessus, $A = \{a, b\}, Q = \{s, p, q\}, T = \{q\}$ et l'on a: $E = \{(s, a, s), (s, b, s), (s, a, p), (p, b, q), (q, a, q), (q, b, q)\}$.

• La relation E est dite *fonctionnelle* ssi elle associe à chaque état et symbole au plus un unique état. L'automate sera alors appelé *déterministe*, et *non-déterministe* sinon. L'automate ci-dessus est non-déterministe (pourquoi ?).

• On dit qu'un automate fini produit ou accepte une phrase sur l'alphabet A si la phrase peut-être obtenue par la concaténation des mots produits lors d'un trajet entre l'état initial et l'un des états terminaux.

• Par extension, un automate accepte un langage L s'il produit toutes les phrases de ce langage et seulement ces phrases.

Notation: si $\mathcal{A}$ est un automate, on note $L(\mathcal{A})$ l'ensemble des phrases acceptées par $\mathcal{A}$, ie le langage accepté par $\mathcal{A}$.

Ex: *abbabaa*, *aaba* sont des phrases acceptées par l'automate ci-dessus

• Une autre manière de considérer les automates est de les voir comme des mécanismes de reconnaissance, plutôt que d'engendrement des phrases (Rabin & Scott 1959). L'automate est une unité de commande à mémoire finie (les états de l'unité de commande), munie d'une tête de lecture qui lit une bande de gauche à droite. L'unité commence dans l'état initial s et scanne la première lettre du mot et passe à l'état suivant en se déplaçant d'un cran à droite;

s'il y a une suite de transitions qui lui permet de scanner successivement toutes les lettres et de s'arrêter, alors il reconnaît le mot, sinon le mot n'appartient pas au langage reconnu par l'automate.

	s		q
	$\downarrow$		$\downarrow$
a	a	b	a
$\uparrow$		$\uparrow$	
s		p	

4.3 Lien avec les grammaires de type 3

• Des grammaires type 3 vers les automates finis:

- on traite les symboles intermédiaires comme les états de l'automate
- on traite les symboles propres comme l'alphabet de l'automate
- si la règle est de type: $A \Rightarrow xB$, on en fait une transition (A, x, B)
- si la règle est de type $A \Rightarrow x$, on en fait une transition (A, x, q) avec q un des états de T.

• Des automates vers les grammaires type 3: construction inverse.

Ex: les règles de réécriture associées à l'automate ci-dessus:

$S \Rightarrow aS$	$P \Rightarrow bQ$	$Q \Rightarrow aQ$
$S \Rightarrow bS$		$Q \Rightarrow bQ$
$S \Rightarrow aP$		$Q \Rightarrow e$

Conséquence: les grammaires linéaires à droite et les automates finis sont équivalents.

4.4 Exercices

1) Considérons l'alphabet: $A = \{$the, old, man, men, is, are, here, and$\}$

a) Construire un automate sur A qui accepte le langage: {the man is here, the men are here}

b) Même chose pour: {the man is here, the men are here, the old man is here, the old men are here, the old old man is here, the old old men are here,...}

c) Construire l'automate qui accepte toutes les phrases de b), plus celles obtenues par la conjonction and.

2) On considère $A = \{a, b\}$

a) Construire un automate qui accepte toute suite qui ne contient que des a (pas la suite vide).

b) Construire un automate qui accepte toute suite qui contient un nombre impair de a, suivi d'un nombre arbitraire (éventuellement nul) de b.

c) Construire un automate qui accepte accepte à la fois les phrases acceptées par l'automate pour a) et celles acceptées par l'automate pour b).

3) Écrire les grammaires de type 3 pour les langages suivants (indice: commencer par chercher l'automate qui correspond, puis en extraire la grammaire):

- $L = \{x$: x contient exactement deux occurrences de a, pas nécessairement contiguës$\}$.
- $L' = \{x$; x contient exactement une occurrence de a, ou exactement une occurrence de b, ou les deux$\}$.

4.5 Non-exprimabilité

On veut montrer que certains langages ne sont pas engendrables par un automate fini. On peut commencer à remarquer que:

Proposition 1
Tout ensemble fini de phrases sur un lexique donné peut être engendré par un automate fini.

Qu'en est-t-il pour un langage infini? Soit $\mathcal{A}$ un automate fini qui engendre un langage infini $L(\mathcal{A})$ (ie un langage avec des phrases arbitrairement longues). Si n est le nombre d'états de $\mathcal{A}$, alors on peut trouver une phrase u de longueur $n+1$. Dans ce cas, l'automate passe forcément au moins deux fois par un même état p (cf la notion de récursivité que nous avons vue):

$$s \overbrace{\xrightarrow{x} p \xrightarrow{y} p \xrightarrow{z}}^{u} t$$

Donc u peut s'écrire xyz, avec $y \neq e$. Mais puisqu'il y a une boucle de p à p qui produit y, cela signifie que toutes les phrases xy^2z, xy^3z, etc., sont également dans $L(\mathcal{A})$. On peut résumer en disant: "tout mot suffisamment long de L contient un facteur itérant" (J. Sakarovitch).

Proposition 2 ("Pumping lemma")
Soit L un langage infini reconnaissable par un automate fini sur un lexique A. Alors on peut trouver trois suites x, y et z de mots de A telles que: y est non vide, et pour tout $n \leq 0$, $xy^nz \in L$

4.6 Exercice

(D'après Partee & al., Schlenker)

1) **Le cas de a^nb^n.** Soit le langage $L = \{a^nb^n; n \geq 1\}$. Utiliser le lemme de pompage pour montrer que ce langage n'est pas reconnaissable par un automate fini. (On raisonne par l'absurde: supposer qu'il l'est, dans ce cas, il existe x, y, z, avec y non vide, tels que $xyz \in L$. Raisonner sur les formes possibles de y et montrer que dans tous les cas, on arrive à une contradiction).

2) **Le langage a^mb^n:**
(i) Montrer que le langage $L' = \{a^mb^n; m, n \geq 1\}$ est reconnaissable par un automate fini.
(ii) Quel lien y a-t-il entre L et L'?

3) **Center embedding**

(i) On considère les phrases suivantes de l'anglais et du français, qui exemplifient la construction dite *center embedding*:
a) A bird died
b) A bird (that) a cat ate died
c) A bird (that) a cat (that) a dog bit ate died.

a') Un oiseau est mort
b') Un oiseau qu'un chat a mangé est mort
c') Un oiseau qu'un chat qu'un chien a mordu a mangé est mort

(ii) Considérons la suite de phrases:

1. A wolf ate
2. A wolf a wolf ate ate
3. A wolf a wolf a wolf ate ate ate

.....

• Quel est le schéma général de ces phrases ?

• On pourrait vouloir conclure de 1) que l'anglais n'est pas engendrable par une grammaire à états finis, car le fragment ci-dessus ne l'est pas. Montrer que ce raisonnement n'est pas concluant (en vous reportant à 2))

4) **Intersection de deux langages.** Montrer que si un langage L est reconnaissable par un automate fini $\mathcal{A}$ et un langage L' est reconnaissable par un automate fini $\mathcal{A}'$, alors l'intersection de L et de L' est reconnaissable par un automate fini.

• Pour cela, on considère l'automate produit $\mathcal{M}$, dont les états sont de la forme (q, q') avec q état de $\mathcal{A}$ et q' état de $\mathcal{A}'$.

• La fonction de transition E sera telle que:
$E = \{((p, p'), a, (q, q'))|(p, a, q) \in E \; et \;\; (q, a, q') \in E'\}$

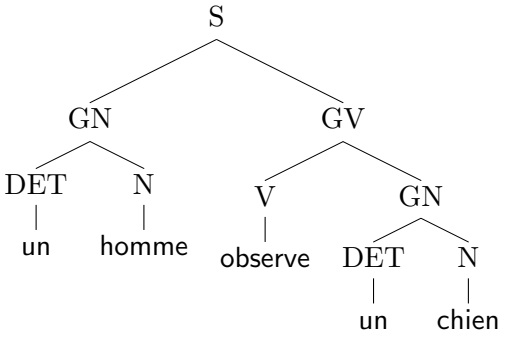
5) Considérer le langage (a wolf)^m(ate)^n. Quelle est son intersection avec l'anglais (ie avec l'ensemble des phrases grammaticales de l'anglais)? Montrer qu'on peut conclure de 4) et de ce qui précède que l'anglais n'est pas engendrable par une grammaire à états finis.

6) Montrer que tout langage $L_k = \{a^n b^n; n \leq k\}$ où k est un entier fixé à l'avance, est engendrable par un automate fini. Décrire l'automate pour $k = 5$. Pourquoi ce résultat ne contredit-il pas celui obtenu en 1) ?

5 Grammaires non-contextuelles

Les grammaires non-contextuelles sont des systèmes de réécriture plus expressifs que les grammaires à états finis, et beaucoup mieux adaptés (bien qu'encore imparfaits) à la description des langues naturelles. Voici un exemple de grammaire non-contextuelle:

S ⇒ GN GV	DET ⇒ un
GN ⇒ DET N	V ⇒ observe
GV ⇒ V GN	N ⇒ chien, homme



• L'intérêt des grammaires non-contextuelles est qu'elles permettent de décrire des constituants syntaxiques : les symboles intermédiaires désignent ici des catégories grammaticales.

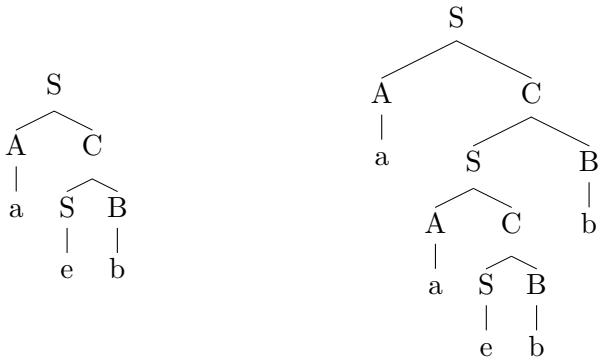
• Comme les grammaires linéaires, les grammaires non-contextuelles correspondent à une classe d'automates. Ces automates sont dits à pile de mémoire (*pushdown automata*). Commes les automates finis, les automates à pile ont un nombre fini d'états, travaillent sur un alphabet fini, et se déplacent de gauche à droite (modèle de Rabin-Scott), mais ils peuvent aussi écrire des symboles auxiliaires sur un registre de mémoire séparé (la pile). L'automate lit, et ajoute ou efface des symboles sur la pile (sur le principe: *last in first out*).

5.1 Exercices

1) Le but de l'exercice est de proposer une grammaire non-contextuelle récursive permettant de dériver les phrases du type: Marie dort, Pierre dort, Marie observe que Pierre dort, Marie observe que Marie dort, Marie observe que Pierre observe que Marie dort, ...

• Outre les catégories déjà vues: S, GN, GV, vous utiliserez les catégories: C, pour complémenteur (que), VI pour verbe intransitif (dort), VT pour verbe transitif (observe), N pour nom (Marie, Pierre). Vous aurez besoin d'une catégorie intermédiaire CP (pour phrase complétive), pour les constituants du type: que Marie dort. Commencer par construire l'arbre syntaxique correspondant à la phrase: Marie observe que Pierre dort. A partir de l'arbre, en inférer les règles de réécriture de la grammaire.

2) Deux grammaires distinctes peuvent engendrer le même langage, on dit alors qu'elles sont équivalentes. Considérer les deux arbres suivants :



• Comparer les phrases engendrées par les deux arbres.
• Proposer un arbre binaire sur le même modèle qui engendre la suite *aaabbb*, la suite a^4b^4, etc.
• Trouver le système de réécriture pour la grammaire associée qui engendre toutes les phrases du type a^nb^n $(n \geq 0)$ à partir d'arbres de cette forme (ie qui ne comporte que des règles du type : $X \Rightarrow YZ$ et $X \Rightarrow x$, où X, Y, Z sont des symboles intermédiaires et x est symbole terminal).
• Une grammaire non-contextuelle où toutes les règles sont de la forme $X \Rightarrow YZ$ et $X \Rightarrow x$ est dite sous *forme normale de Chomsky*. En conclure que le système obtenu normalise la grammaire de l'exemple (8).

5.2 Perspectives

• Pouvoir expressif des grammaires non-contextuelles ? Limité aussi: cf. le langage $a^nb^nc^n$.
• Hiérarchie de grammaires/d'automates et limitations computationnelles
• Machines de Turing: automates plus puissants encore; peuvent lire et écrire, aller à gauche et à droite.

Références bibliographiques

N. Chomsky (1967), *Le langage et la pensée*, tr. fr. Payot.
B.H. Partee, A. ter Meulen, R.E. Wall, *Mathematical Methods in Linguistics*, Kluwer 1990 [c. 16-18].
L.F.T. Gamut (1991), *Logic, Language and Meaning*, vol 1. *Introduction to Logic*, Chicago [c. 7].
J. Sakarovitch (2003), *Éléments de théorie des automates*, Vuibert Informatique [ouvrage avancé, approche plus algébrique].
P. Schlenker (2004-2005), Notes de cours d'introduction à la linguistique, ENS 2004 et 2005, `http://www.linguistics.ucla.edu/people/schlenker/`
M. Sipser (1996), *Introduction to the Theory of Computation*, PWS Publishing Company.