

Université Bordeaux 1
Bruno Mery
Master Informatique, 2ème année

Mémoire de Recherche

Année 2005-2006

Grammaires légèrement contextuelles
pour l'analyse syntaxique
du langage naturel

Sous la direction de :
Maxime Amblard
Irène Durand
Christian Retoré

Table des matières

1	Introduction	5
2	Aperçu de la situation des divers formalismes utilisés	8
2.1	Préalables et place relative des formalismes étudiés	8
2.1.1	Hiérarchie de Chomsky, langages réguliers, algébriques et contextuels	9
2.1.2	Langages légèrement contextuels	10
2.2	Hiérarchie des formalismes	12
2.2.1	Classes voisines des TAG	12
2.2.2	Classes voisines des Grammaires Minimalistes	13
3	Définitions formelles et exemples	15
3.1	Grammaires à adjonction d'arbres	15
3.1.1	Grammaire à adjonction d'arbres, à composantes multiples	18
3.2	Grammaires hors contexte généralisées et restrictions	18
3.2.1	Grammaires hors contexte généralisées	19
3.2.2	Grammaires hors contexte k-multiples parallèles	20
3.2.3	Grammaires hors contexte k-multiples	22
3.2.4	Systèmes de réécriture hors contexte linéaires	23
3.3	Grammaires à concaténation d'intervalles	23
3.3.1	Grammaires à concaténation d'intervalles, simples et positives	25
3.4	Grammaires Minimalistes	26
3.4.1	Structure d'arbres ordonnés avec projection	26
3.4.2	Traits et lexiques	27
3.4.3	Grammaires Minimalistes classiques	29
3.4.4	Grammaires Minimalistes augmentées	32

4	Équivalences et conversions des formalismes	34
4.1	Variantes des PMCFG, MCFG et RCG	34
4.1.1	PMCFG et RCG	34
4.1.2	LCFRS et sPRCG	34
4.1.3	Forme normale des MCFG	35
4.2	Équivalence MG-MCFG et analyse des grammaires minimalistes	35
4.2.1	Procédure de conversion des grammaires minimalistes en MCFG	36
4.2.2	Exemple de conversion	39
4.2.3	Complexité de la conversion	43
4.2.4	Analyse des grammaires minimalistes	45
5	Conclusion	48
A	Glossaire des acronymes de formalismes	51
B	Références bibliographiques et Internet	54
C	Remerciements	58

Table des figures

2.1	Place des langages légèrement contextuels dans la hiérarchie de Chomsky	11
2.2	Hiérarchie de quelques formalismes légèrement contextuels, plus proches des langages algébriques	12
2.3	Hiérarchie de quelques formalismes légèrement contextuels, plus proches des grammaires minimalistes	13

Chapitre 1

Introduction

La langue est une raison humaine qui a ses raisons, et que l'homme ne connaît pas.

C. Lévi-Strauss

Bien que des études sur la nature des langues humaines aient été conduites tout au long de l'Histoire, ce n'est que récemment que la linguistique s'est intéressée aux structures pouvant représenter ces langues. Ces travaux sont devenus d'une importance considérable lorsque les informaticiens, tentant de réaliser un système de traduction automatique, se heurtèrent à l'incommensurable complexité des langues. Il est vite apparu que, pour rendre compte de l'ensemble des phénomènes linguistiques, de nouveaux formalismes devaient être construits.

La compréhension par un ordinateur du sens d'un énoncé arbitraire est un problème qu'il sera sans doute très long et difficile de résoudre entièrement : on estime qu'il est *IA-complet*, c'est-à-dire que sa solution implique la création d'un système d'une intelligence comparable à l'humain (plusieurs des progrès réalisés ces derniers temps sont d'ailleurs basés sur l'analyse des processus cognitifs humains). Cependant, l'analyse automatisée de certains éléments de syntaxe, de grammaire ou de sémantique des langues a, aujourd'hui, de nombreuses applications : recherche d'informations dans de grandes bases de données, traduction de textes spécialisés (telles les notices techniques) et aide à la traduction, correction des fautes de frappe et (dans une certaine mesure) de fautes d'or-

thographe, interprétation de questions, génération de textes aléatoires (à la manière de la littérature combinatoire de Raymond Queneau), sont autant de champs pour lesquels la recherche en linguistique computationnelle et en traitement automatique des langues a permis des progrès rapides.

De nombreux formalismes mathématiques, nés de l'informatique ou de la linguistique, ont ainsi été théorisés. Leur objectif ultime est de décrire l'ensemble des processus linguistiques mis en jeu dans la communication humaine, tout en restant assez simples à mettre en œuvre et efficaces pour une utilisation quotidienne. Nous exposerons ici les formalismes existants qui apparaissent aujourd'hui les plus à même de remplir cet objectif, en nous limitant à la capacité générative – celle de pouvoir engendrer, de façon structurée, les langages recherchés, sans s'intéresser aux problématiques liées à d'autres aspects du traitement automatique des langues, comme la génération de formules logiques associées aux phrases, la mise en relation avec des bases de données sémantiques ou pragmatiques, qui constituent autant de sujets de recherche à part entière.

D'un point de vue linguistique, la structuration de la phrase donnée par ces formalismes correspond à la *syntaxe* de la phrase, c'est-à-dire à son organisation par rapport aux règles grammaticales de la langue. Le but est de modéliser la *compétence* du locuteur, c'est-à-dire sa capacité à analyser la syntaxe d'une phrase arbitrairement longue, connaissant les règles de grammaire et le vocabulaire employés. Par opposition, la prise en compte de sa *performance*, soit le fait qu'en temps normal, une phrase comporte un nombre fini et relativement limité de mots (quelques dizaines au plus) et de constructions syntaxiques (par exemple, une phrase contient rarement plus de trois subordinées relatives imbriquées les unes dans les autres), permettrait de rendre compte beaucoup plus facilement des langues telles qu'elles sont couramment parlées, mais non des processus qui permettent de les appréhender...

Ainsi, les *grammaires algébriques* ou *hors contexte*, analogues aux règles de grammaires enseignées en Primaire¹, constituent un formalisme mathématiquement simple et très efficace, qui peut d'ailleurs être utilisé pour un langage restreint (comme un langage de programmation), mais se sont cependant révélées insuffisantes pour les langues humaines.

À l'inverse, d'autres formalismes peuvent exprimer tous les langages

¹De la forme : un *groupe nominal* peut être un *nom propre*, un *déterminant* suivi d'un *nom commun*, ou bien un *déterminant* suivi d'un *adjectif antéposé* et d'un *nom commun*...

récurivement énumérables, mais ne sont ni assez précis, car ils ne permettent pas de se circonscrire aux phénomènes linguistiques voulus et génèrent des langages bien plus complexes que les langues humaines, ni assez performants, le problème de l'appartenance d'une phrase à un langage issu de ces formalismes étant, au mieux, NP-complet.

Un des buts des théoriciens a été, au fil du temps, de trouver un formalisme approchant le plus possible d'une borne minimale, c'est-à-dire qui prenne en compte tous les phénomènes linguistiques, sans plus.

Des études en linguistique et en psychologie ont permis de dégager deux conjonctures sur ces formalismes. D'une part, ils doivent permettre l'analyse des phrases en temps raisonnable, sans quoi les locuteurs ne pourraient comprendre une phrase de plus de quelques mots assez rapidement pour pouvoir tenir une conversation (ce qui est le plus souvent interprété comme un temps d'analyse, au pire, polynomial). D'autre part, il doit exister un algorithme permettant l'apprentissage d'un langage donné, engendré par l'un de ces formalismes, à partir d'exemples positifs – c'est-à-dire que l'on doit pouvoir dégager une description du langage à partir d'un certain nombre de phrases correctement construites, comme le font les jeunes enfants.

Une caractérisation, proposée par Joshi, nomme les formalismes pouvant répondre à ce problème, ainsi que les langages qu'ils engendrent, *légèrement contextuels*. Nous allons présenter cette caractérisation, ainsi que la place des langages engendrés par de tels formalismes, puis les formalismes eux-mêmes plus en détail, et étudier les équivalences existantes entre eux. Nous nous intéresserons particulièrement aux *grammaires minimalistes*, résultantes de la synthèse d'hypothèses linguistiques portant sur l'universalité du langage, et aux *grammaires à concaténation d'intervalles*, possédant d'intéressantes propriétés formelles, ainsi qu'à plusieurs formalismes intermédiaires.

Chapitre 2

Aperçu de la situation des divers formalismes utilisés

Avoir un système borne son horizon ; n'en avoir pas est impossible. Le mieux est d'en posséder plusieurs.

R. Queneau

2.1 Préalables et place relative des formalismes étudiés

Nous allons ici présenter la place des formalismes que nous nous proposons de présenter, leurs relations entre eux et avec d'autres formalismes classiques. Ici, ils seront classés selon leur *pouvoir d'expression*, c'est-à-dire la classe de langages qu'ils sont capables d'engendrer. Nous nous intéressons ici en effet à la modélisation de la *compétence* du locuteur : la capacité, pour un humain connaissant le vocabulaire et les règles grammaticales d'une langue, de produire des phrases correctes (y compris des phrases qu'il n'a jamais entendues auparavant), et de pouvoir déterminer si une phrase donnée (qu'il peut, là aussi, ne pas connaître) est grammaticalement correcte ou non. Comme nous l'avons dit précédemment, nous ne nous attacherons pas à sa *performance*.

2.1.1 Hiérarchie de Chomsky, langages réguliers, algébriques et contextuels

Noam Chomsky, linguiste réformateur et agitateur politique américain très influent depuis 1955, est le créateur de la théorie des *grammaires génératives*. Les différentes propriétés de ces grammaires, qui engendrent des langages très divers selon les contraintes qu'on leur impose, lui ont permis de définir une hiérarchie, qui sert de cadre de référence pour tous les autres formalismes.

Une grammaire générative est définie par un ensemble N de symboles non-terminaux (notés A, B, \dots), un ensemble T de symboles terminaux, représentant ici les mots ou idiomes de la langue¹ (notés a, b, \dots – T est également l'alphabet sur lequel sera écrit le langage engendré), et un ensemble de règles de la forme $\alpha \rightarrow \beta$, où $\alpha, \beta \in (N \cup T)^*$. Étant donné un *axiome* $S \in N$, le langage engendré par la grammaire sera l'ensemble des chaînes γ de terminaux tels que $S \rightarrow^* \gamma$.

Par exemple, les règles de français décrites par *un groupe nominal (NP) peut être un nom propre (N), un déterminant (D) suivi d'un nom commun (NC), ou bien un déterminant suivi d'un adjectif antéposé (AA) et d'un nom commun* seront ainsi exprimées dans des règles de grammaires génératives :

$$\begin{aligned} NP &\rightarrow N \\ NP &\rightarrow D \quad NC \\ NP &\rightarrow D \quad AA \quad NC \end{aligned}$$

Les langages de la hiérarchie de Chomsky forment quatre classes, dites "type-0", "type-1", "type-2" et "type-3", avec :

$$\text{type-3} \subsetneq \text{type-2} \subsetneq \text{type-1} \subsetneq \text{type-0}$$

Il existe également des langages non récursivement énumérables, qui ne possèdent aucune des propriétés caractérisant ces classes, et sont peu étudiés².

Ces quatre classes sont définies comme suit :

¹Selon l'usage, ils peuvent aussi représenter des lettres, ou, par exemple, dans des études morphologiques, des phonèmes ou morphèmes élémentaires, permettant de former les mots par flexion, etc. ...

²De fait, les langages de type 0 étant *Turing-complets*, et regroupant donc tous les langages pouvant être décrits par un dispositif ou formalisme algorithmique arbitraire, nous ne pouvons avoir de description utile des langages non récursivement énumérables. La

type-0 : L'ensemble des langages *récursivement énumérables*. De nombreux formalismes permettent de les construire, dont les machines de Turing et les grammaires génératives non restreintes. Cet ensemble est cependant trop important pour un traitement automatisé, et l'analyse de ces langages pose un problème au moins NP-complet.

type-1 : Les langages *contextuels*. Par rapport aux classes de langages plus restreintes, les langages contextuels supposent la connaissance de données extérieures au terme en cours d'analyse³.

type-2 : Les langages *algébriques*, ou *hors contexte*. Ces langages sont engendrés et caractérisés par l'ensemble des *Grammaires hors contexte* (*Context-Free Grammars* ou CFG), très étudiées pour leurs propriétés (un mot d'un langage algébrique quelconque, de grammaire connue, est notamment analysable en $O(n^3)$, pour un mot ou une phrase de n symboles).

type-3 : Les langages *réguliers*, construits et reconnus par des automates à états finis, ou de manière équivalente par des expressions régulières. Ces formalismes sont utiles pour des applications très spécialisées, car très performants, mais sont trop restreints pour la plupart des applications du traitement automatique des langues.

2.1.2 Langages légèrement contextuels

Individuellement, les phénomènes linguistiques sont, pour la plupart, représentables par des langages algébriques. Cependant, de nombreux travaux, dont [Stu85], montrent que ces langages n'ont pas le pouvoir d'expression nécessaire pour représenter une langue dans son ensemble – en particulier, les phénomènes de références croisées leur échappent⁴. Divers arguments, dont la nécessité d'une analyse rapide et de leur apprenabilité, donnent à penser que les langues humaines sont incluses dans les langages

seule preuve qui puisse être donnée de leur existence est que leur non-existence aboutirait à une contradiction. On peut donc aisément comprendre que cette classe de langages, en particulier, n'intéresse que modérément les linguistes...

³Une *grammaire contextuelle* serait analogue à des règles de construction de cette forme : un *groupe verbal*, quand il est suivi d'un *complément d'objet*, peut être constitué d'un *groupe sujet* et d'un *verbe transitif*. La présence du complément d'objet constitue ici une information extérieure au groupe verbal, faisant partie de son *contexte*.

⁴Ainsi, en français, une phrase de type $A_1, A_2 \dots$ et A_n , qui habitent respectivement à $B_1, B_2 \dots$ et B_n , correspond, dans la forme, à $\{w^2\}$, qui n'est pas un langage algébrique. De nombreuses études ont été menées pour des structures de phrases non algébriques plus fréquentes en néerlandais, ou, comme [Stu85], en suisse allemand.

contextuels. Dans [Jos85], Joshi propose une caractérisation pour ce qu'il nomme *langages légèrement contextuels* (*Mildly Context-Sensitive Languages* ou MCSL), visant à approcher la plus petite classe possible de langages incluant tous les phénomènes linguistiques. Cette classe regroupe les langages possédant au moins les propriétés suivantes, présentées de manière succincte dans [Jos] :

1. Les langages algébriques sont strictement inclus dans MCSL.
2. Tout langage de MCSL peut être analysé en temps polynomial.
3. Les langages de MCSL sont capables de représenter certains phénomènes linguistiques. L'auteur laisse une marge d'appréciation quant à ces phénomènes, et suggère de considérer les références croisées de certaines langues germaniques, en limitant leur nombre, par exemple.
4. Les langages de MCSL sont dits *constamment croissants*.

Cette définition est volontairement ambiguë. Ainsi, on considère habituellement les grammaires d'adjonction d'arbres (*Tree Adjoining Grammars* ou TAG) et les grammaires hors-contexte multiples (*Multiple Context-Free Grammars* ou MCFG) comme engendrant des langages de MCSL.

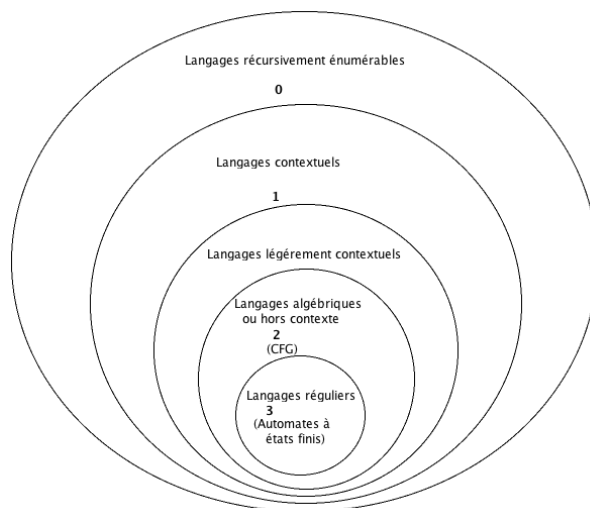


FIG. 2.1 – Place des langages légèrement contextuels dans la hiérarchie de Chomsky

2.2 Hiérarchie des formalismes

Nous présentons ici la place relative de quelques formalismes, parmi les plus utilisés. Ceux qui nous intéressent seront définis et détaillés par la suite.

2.2.1 Classes voisines des TAG

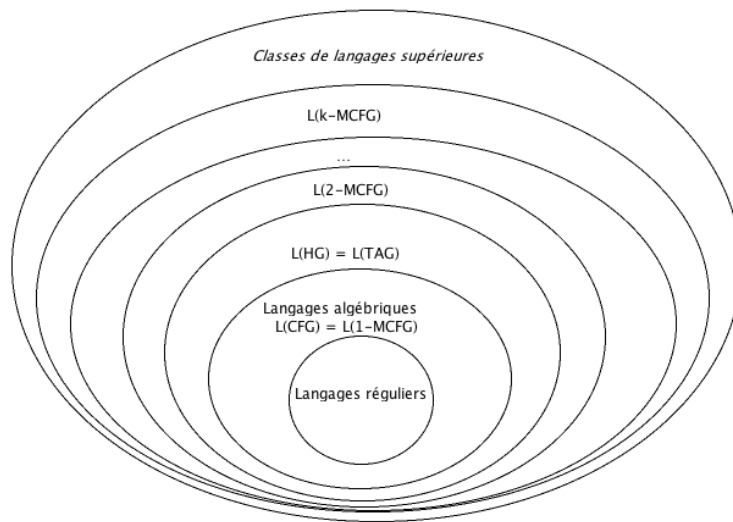


FIG. 2.2 – Hiérarchie de quelques formalismes légèrement contextuels, plus proches des langages algébriques

Les grammaires non-contextuelles k -multiples (k -MCFG) sont des formalismes proches des grammaires algébriques. Par définition, les 1 -MCFG sont équivalentes aux CFG, et l'ensemble des $k+1$ -MCFG contient strictement les k -MCFG. Les définitions et preuves de ces équivalences et inclusions sont données dans [SMFK91].

$$\begin{aligned} L(CFG) = L(1-MCFG) \subsetneq L(2-MCFG) \subsetneq \dots \\ \subsetneq L(k-MCFG) \subsetneq L(k+1-MCFG) \subsetneq \dots \end{aligned}$$

Les grammaires de tête (*Head Grammars*, HG) et les grammaires d'adjonction d'arbres (*Tree Adjoining Grammars*, TAG) sont des formalismes couramment étudiés, dont les classes de langages de chaînes sont équivalentes et strictement incluses dans celle des 2-MCFG. Ces équivalences et inclusions sont également prouvées dans [SMFK91].

$$L(CFG) \subsetneq L(HG) = L(TAG) \subsetneq L(2-MCFG)$$

2.2.2 Classes voisines des Grammaires Minimalistes

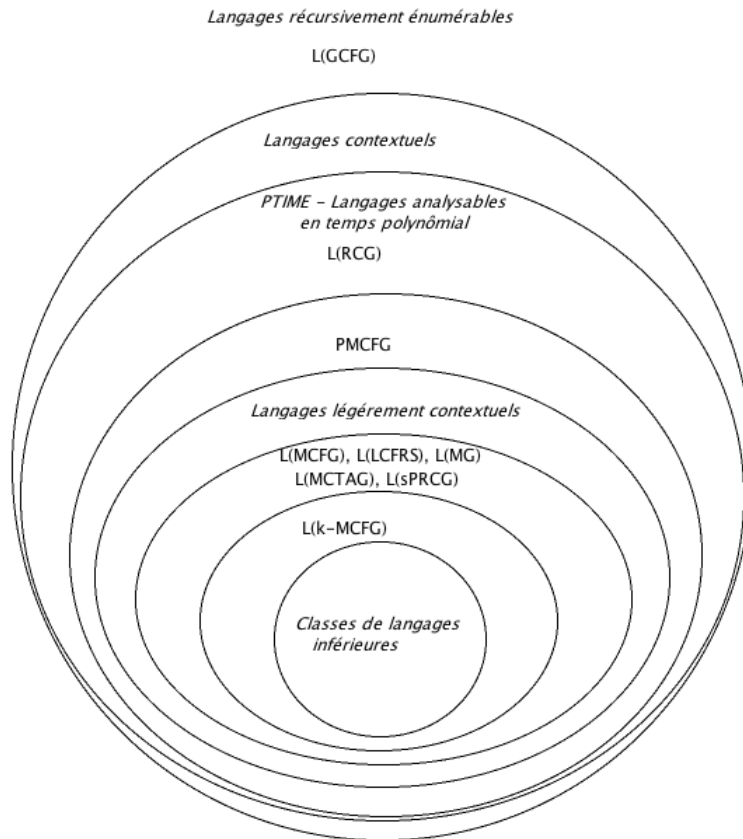


FIG. 2.3 – Hiérarchie de quelques formalismes légèrement contextuels, plus proches des grammaires minimalistes

Les MCFG (k-MCFG pour un k arbitraire), les systèmes de réécriture

hors-contexte linéaires (*Linear Context-Free Rewriting Systems*, LCFRS), les grammaires d'adjonction d'arbres à composants multiples (*Multiple Component Tree Adjoining Grammars*, MCTAG), les grammaires à concaténation d'intervalles simples et positives (*simple Positive Range Concatenation Grammars*, sPRCG), ainsi que les grammaires minimalistes (*Minimalist Grammars*, MG), sont des formalismes générant une même classe de langage. [SMFK91] démontre l'équivalence des MCFG et des LCFRS, [vV96] précise leur équivalence avec les MCTAG, [Bou98] prouve l'équivalence des LCFRS et des sPRCG, et [Mic01d], associé à [Mic01a] démontre l'équivalence des LCFRS avec les grammaires minimalistes. Les MG, LCFRS et sPRCG nous intéresseront particulièrement. Tous les langages de cette classe sont considérés comme légèrement contextuels, étant analysables en temps polynomial et constamment croissants.

$$\begin{aligned} & \bigcup_{k \in \mathbb{N}} (L(k\text{-MCFG})) = L(\text{MCFG}) \\ & = L(\text{LCFRS}) = L(\text{MCTAG}) = L(\text{sPRCG}) = L(\text{MG}) \subset \text{MCSL} \end{aligned}$$

Les grammaires hors contexte multiples parallèles (PMCFG) ont un pouvoir plus grand que les MCFG, comme démontré par [SMFK91].

$$L(\text{MCFG}) \subsetneq L(\text{PMCFG})$$

Les grammaires à concaténation d'intervalles (RCG), généralisation des sPRCG, ont pour particularité d'engendrer, non seulement les langages légèrement contextuels, mais aussi exactement tous les langages analysables en temps polynomial (avec exposant arbitraire). Les langages humains étant considérés comme compréhensibles (c'est-à-dire facilement analysables), ils sont inclus dans cette classe de langages. La définition de ces formalismes, ainsi que la caractérisation formelle de leur pouvoir d'expression, sont donnés dans [Bou98].

$$\text{MCSL} \subsetneq L(\text{PMCFG}) \subsetneq L(\text{RCG}) = \text{PTIME} \subsetneq \text{CSL}$$

Les PMCFG et RCG génèrent plus que les langages légèrement contextuels, pouvant exprimer $\{a^{(2^n)}\}$, par exemple.

Enfin, les grammaires hors contextes généralisées (GCFG), permettant de formaliser les MCFG, engendrent exactement l'ensemble des langages récursivement énumérables, dont les langages contextuels. Cette caractérisation est donnée dans [SMFK91].

$$\text{MCSL} \subsetneq \text{CSL} \subsetneq \text{REL} = L(\text{GCFG})$$

Chapitre 3

Définitions formelles et exemples

L'homme est à la recherche d'un nouveau langage auquel la grammaire d'aucune langue n'aura rien à dire.

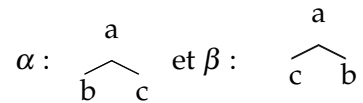
G. Apollinaire

3.1 Grammaires à adjonction d'arbres

Les *grammaires à adjonction d'arbres* (*Tree Adjoining Grammars* ou TAG) ont été proposées par Joshi dans [JLT75] et sont un des premiers exemples de formalismes légèrement contextuels. Le principe est de visualiser les relations syntaxiques, ainsi que la structure des éléments considérés, sur des arbres de la forme :



Tous les arbres sont ici considérés comme ordonnés, de gauche à droite, de façon à ce que

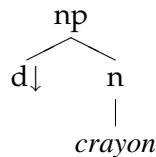


soient deux arbres différents.

Formellement, les TAG sont définies (ces définitions et exemples sont tirés de [Van93]) par un ensemble d'*arbres élémentaires*, dont les :

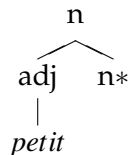
Arbres initiaux : Un arbre initial est un arbre dont les nœuds internes sont étiquetés par des *symboles non-terminaux*¹. Les feuilles sont étiquetées, soit par des terminaux², soit par des non-terminaux marqués par un *marqueur de substitution*, ↓.

Cet arbre définit ainsi une entrée lexicale, *crayon*, de type *syntagme nominal* (np) :



Arbres auxiliaires : Un arbre auxiliaire est défini comme un arbre initial, dont une ou plusieurs des feuilles sont étiquetées par le même non-terminal que sa racine. L'une d'entre elles est appelée *pied* de l'arbre, et est particularisée par un marqueur *.

Cet arbre définit ainsi la structure d'un nom pouvant comporter l'adjectif antéposé *petit* :



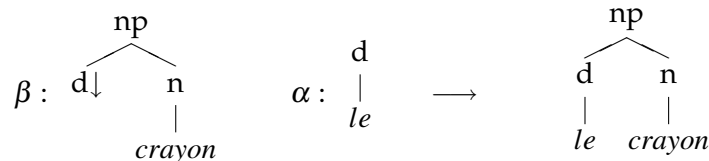
¹Représentant, en général, les catégories grammaticales ou syntaxiques, comme *N*, *V*, ou bien *adj*, *vp*...

²Mots ou groupes de mots de la phrase, en général.

Les opérations de *substitution* et d'*adjonction* permettent d'obtenir, à partir des arbres élémentaires, les *arbres dérivés* de la grammaire. Elles sont définies comme suit :

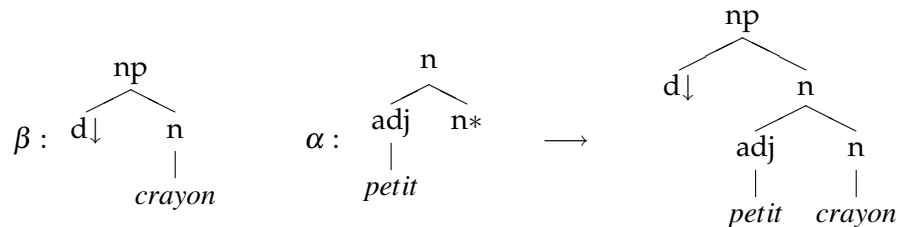
Substitution : Un arbre (initial ou dérivé) α de racine N peut être substitué dans un arbre β en remplaçant une feuille étiquetée $N \downarrow$ dans β par l'arbre α .

Par exemple, les deux arbres suivants peuvent se combiner par substitution :



Adjonction : Un arbre auxiliaire α de racine (et donc pied) étiquetée par N peut être adjoint à un arbre β , dans un nœud n de β étiqueté par n . Le nœud n est alors remplacé par α , et l'ancien sous-arbre de β enraciné en n est alors enraciné à la feuille correspondant au pied de α .

Par exemple, les deux arbres suivants peuvent se combiner par adjonction :



Une TAG donnée, ayant pour symbole initial S , a alors pour langage d'arbres l'ensemble des arbres, de racine S , dérivables à partir de ses arbres élémentaires, ne possédant plus que des symboles terminaux sur leurs feuilles. Son langage à proprement parler est l'ensemble des chaînes qui sont formées par concaténation de ces terminaux, lus dans l'ordre usuel.

Les TAG sont des formalismes légèrement contextuels, permettant l'analyse syntaxique de phrases de longueur n en temps $O(n^6)$.

Une application des TAG est la *lexicalisation* de grammaires hors contexte, c'est-à-dire la possibilité de représenter les règles de CFG avec au plus un terminal associé à une règle pour toute CFG, sans changer la structure syntaxique qu'elle modélise. Une grammaire lexicalisée est utile pour réaliser un dictionnaire avec pour entrée le terminal associé à une règle, par exemple...

3.1.1 Grammaire à adjonction d'arbres, à composantes multiples

Les *grammaires à adjonction d'arbres, à composantes multiples* (*Multiple Component Tree Adjoining Grammars* ou MCTAG) sont une variante des TAG qui peut sembler triviale : pour les arbres auxiliaires d'une MCTAG, on autorise plusieurs pieds (donc plusieurs façons de réaliser une adjonction). Cette variante augmente considérablement la classe de langages engendrés.

3.2 Grammaires hors contexte généralisées et restrictions

[SMFK91] présente en détail plusieurs formalismes introduits plus haut, (GCFG, PMCFG, MCFG et LCFRS), que nous allons résumer ici. Ils généralisent la notion de *grammaire algébrique* (dite aussi *hors contexte* ou *non contextuelle*), représentant les grammaires génératives dont les règles sont du type :

$$A \rightarrow \alpha B$$

où A, B sont des symboles non terminaux et α une suite de symboles terminaux (lettres).

Les *grammaires hors contexte généralisées* (*generalized context-free grammars* ou GCFG), premier de ces formalismes, utilisent des règles analogues :

$$A_0 \rightarrow f[A_1, \dots, A_q]$$

$$A_i \rightarrow \alpha_i$$

où les A_i sont des symboles non terminaux, les α_i des suites de terminaux et f une fonction. Les propriétés de ces grammaires, détaillées ci-

après, permettent l'expression des langages de type-0, et sont donc trop générales. L'introduction de contraintes, portant sur les règles et les fonctions utilisées, permet de définir les *grammaires hors contexte k-multiples parallèles* (*parallel multiple context-free grammars* ou PMCFG), les *grammaires hors contextes k-multiples* (*multiple context-free grammars* ou MCFG), et les *systèmes de réécriture hors contexte linéaires* (*linear context-free rewriting systems* ou LCFRS).

3.2.1 Grammaires hors contexte généralisées

Une GCFG G est définie par :

$$G = (N, O, F, P, S)$$

où N est un ensemble fini de symboles non-terminaux (A_0, A_1, \dots) , dont le symbole initial S . Les éléments de O sont des tuples de chaînes sur un ensemble de symboles terminaux T , ceux de F sont des fonctions partielles f de $O^q \rightarrow O$ (pour un entier q donné), et les règles de P sont de la forme :

$$A_0 \rightarrow f[A_1, \dots, A_q]$$

Il peut y avoir plusieurs occurrences d'un non-terminal A_i donné en partie droite. Si $q = 0$, la règle est terminale ($A \rightarrow f[]$ et $f[] = \alpha$ équivaut à $A \rightarrow \alpha$, pour une chaîne $\alpha \in O$ de (tuples de) terminaux).

Le langage de G pour un non-terminal donné A , $L_G(A)$, est défini comme le plus petit ensemble pour lequel :

- Si $A \rightarrow \theta \in P$, où $\theta \in O$, (cas d'une règle terminale), alors $\theta \in L_G(A)$.
- Si on a :
 - $\theta_i \in L_G(A_i)$ pour $1 \leq i \leq q$,
 - $A \rightarrow f[A_1, \dots, A_q]$ est une règle de P , et
 - $f[\theta_1, \dots, \theta_q]$ est défini,
 alors $f[\theta_1, \dots, \theta_q] \in L_G(A)$.

Et on définit $L_G = L_G(S)$ le langage produit par G .

3.2.2 Grammaires hors contexte k–multiples parallèles

Les *grammaires hors contexte k–multiples parallèles* sont des GCFG sur lesquelles on impose certaines propriétés structurelles.

Une GCFG $G = (N, O, F, P, S)$ est une m –PMCFG si elle satisfait :

- $O = \bigcup_{i=1}^m (T^*)^i$, où T est un ensemble de symboles (terminaux), disjoint de N . Autrement dit, O est l'ensemble des tuples de chaînes de symboles de T , d'arité au plus m .
- On pose $a(f)$ le nombre d'arguments de $f \in F$. On définit également $r(f)$ et $d_i(f)$, pour $1 \leq i \leq a(f)$, entiers positifs inférieurs à m , de façon à ce que f soit une fonction de :

$$(T^*)^{d_1(f)} x (T^*)^{d_2(f)} x \dots (T^*)^{d_{a(f)}(f)} \rightarrow (T^*)^{r(f)}$$

Ainsi, chaque argument de f est un tuple d'arité fixe et connue, de même que son résultat en partie droite, portant sur des chaînes de symboles de T .

De plus, chaque composante de la partie droite de f doit être une concaténation de chaînes constantes de T données et d'éléments des arguments de f , par exemple, avec des variables z_i et des constantes α, β :

$$f[\dots, (z_1, z_2, \dots), \dots] = (\dots, \alpha z_1 \beta z_3, \dots)$$

- Pour tout $A \in N$, on définit $d(A)$ la *dimension* du non-terminal A . On impose ensuite la condition suivante sur les fonctions utilisées : si $A \rightarrow f[A_1, \dots, A_{a(f)}] \in P$, alors $r(f) = d(A)$ et, pour $1 \leq i \leq a(f)$, on a $d_i(f) = d(A_i)$. Les symboles non-terminaux ont donc une arité (dimension) fixée, correspondant à celle des arguments des fonctions qui peuvent leur être appliquées.
- Avec ces conventions, on impose $d(S) = 1$.

Les k –PMCFG imposent donc une arité à chacun de leurs non-terminaux. L'arité maximale de ces symboles est k , et elle détermine, dans une certaine mesure, les capacités génératives de la grammaire, et on a :

$$L(k\text{-PMCFG}) \subsetneq L(k+1\text{-PMCFG}) \subsetneq \dots \subsetneq \bigcup_{k \in \mathbb{N}} L(k\text{-PMCFG}) = L(\text{PMCFG})$$

On peut ainsi définir, à l'aide d'une 1–PMCFG, le langage $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$:

$$\begin{aligned}
T &= \{0, 1\}, N = \{A, S\}, O = T^* \\
A &\rightarrow 0 \mid 1 \mid g_1[A, A] \\
S &\rightarrow g_2[A] \\
g_1[x, y] &= xy \\
g_2[x] &= xx
\end{aligned}$$

Ici, A est de dimension $d(A) = 1$ (A ne se dérive qu'en éléments de T : 0 ou 1), g_1 est de rang 2 (prenant deux arguments, tous deux de dimension 1), g_2 est de rang 1. Comme la dimension maximale des symboles utilisés est de 1, il s'agit bien d'une 1-PMCFG, et $O = \bigcup_{i=1}^1 (T^*) = T^*$. Une séquence de dérivations produisant, à l'aide de cette grammaire, un mot du langage, pourrait s'écrire par exemple :

$$\begin{aligned}
S &\rightarrow g_2[A] \\
S &\rightarrow g_2[g_1[A, A]] \\
S &\rightarrow g_2[g_1[g_1[A, A], g_1[A, A]]] \\
S &\rightarrow g_2[g_1[g_1[0, g_1[A, A]], g_1[g_1[A, A], 1]]] \\
S &\rightarrow g_2[g_1[g_1[0, g_1[0, 1]], g_1[g_1[1, 0], 1]]] \\
S &\rightarrow g_2[g_1[g_1[0, 01], g_1[10, 1]]] \\
S &\rightarrow g_2[g_1[001, 101]] \\
S &\rightarrow g_2[001101] \\
S &\rightarrow 001101001101
\end{aligned}$$

On peut également définir, toujours à l'aide d'une 1-PMCFG, le langage $\{\alpha^{(2^n)}\}$ (ici, pour $\alpha = a$) :

$$\begin{aligned}
T &= \{a\}, N = \{S\}, O = T^* \\
S &\rightarrow a \mid g_1[S] \\
g_1[x] &= xx
\end{aligned}$$

1

Ainsi que le langage $\{a^{(n^2)} \mid n > 0\}$, avec la 2-PMCFG suivante :

$$\begin{aligned}
T &= \{a\}, N = \{A, S\}, O = T^* \cup (T^*)^2 \\
A &\rightarrow (\varepsilon, \varepsilon) \mid g_1(A) \\
S &\rightarrow g_2[A] \\
g_1[(x_1, x_2)] &= (ax_1, ax_1^2x_2) \\
g_2[(x_1, x_2)] &= ax_1^2x_2
\end{aligned}$$

Ici, le symbole A est de dimension 2, et les fonctions utilisées sont de rang 1, mais portent sur des couples de chaînes. Une séquence de dérivations pourrait s'écrire :

$$\begin{aligned}
S &\rightarrow g_2[A] \\
S &\rightarrow g_2[g_1[A]] \\
S &\rightarrow g_2[g_1[g_1[A]]] \\
S &\rightarrow g_2[g_1[g_1[g_1[A]]]] \\
S &\rightarrow g_2[g_1[g_1[g_1[(\varepsilon, \varepsilon)]]]] \\
S &\rightarrow g_2[g_1[g_1[(a, a)]]] \\
S &\rightarrow g_2[g_1[(aa, a^4)]] \\
S &\rightarrow g_2[(aaa, a^9)] \\
S &\rightarrow a^{16}
\end{aligned}$$

3.2.3 Grammaires hors contexte k-multiples

Une *grammaire hors contexte k-multiple* (*k-multiple context-free grammar*, *k-MCFG*) est une *k-PMCFG* pour laquelle on impose la *condition d'inhibition de copie*. Cette condition consiste simplement à interdire à une variable d'une fonction f donnée (c'est-à-dire un des éléments d'un argument de f) d'apparaître plus d'une fois en partie droite de f .

On peut ainsi redéfinir le langage (déjà décrit par une 1-MCFG) $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$, cette fois-ci à l'aide d'une 2-MCFG :

$$\begin{aligned}
T &= \{0, 1\}, N = \{A, S\} \\
A &\rightarrow (0, 0) \mid (1, 1) \mid g_0[A] \mid g_1[A] \\
S &\rightarrow g_2[A] \\
g_0[(x, y)] &= (0x, 0y) \\
g_1[(x, y)] &= (1x, 1y) \\
g_2[(x, y)] &= xy
\end{aligned}$$

Pour un m donné, le langage $\{a_1^n a_2^n \dots a_{2m}^n \mid n \geq 0\}$ peut être exprimé par une m -MCFG sur le modèle qui suit :

$$\begin{aligned}
T &= \{a_1, \dots, a_{2m}\}, N = \{A, S\} \\
A &\rightarrow (\varepsilon, \varepsilon, \dots, \varepsilon) \mid g_1[A] \\
S &\rightarrow g_2[A] \\
g_1[(x_1, x_2, \dots, x_m)] &= (a_1 x_1 a_2, a_3 x_2 a_4, \dots, a_{2m-1} x_m a_{2m}) \\
g_2[(x_1, x_2, \dots, x_m)] &= x_1 x_2 \dots x_m
\end{aligned}$$

3.2.4 Systèmes de réécriture hors contexte linéaires

Bien qu'introduits de façon indépendante des GCFG, PMCFG et MCFG, les *systèmes de réécriture hors contexte linéaires* (*Linear Context-Free Rewriting Systems*, LCFRS) constituent également une restriction de cette classe de formalisme. Ainsi, un k -LCFRS est une k -MCFG pour laquelle on impose la *condition de non-effacement*, qui consiste à imposer, pour chaque variable d'un argument donné d'une fonction f , d'apparaître exactement une fois (et non plus au moins une fois) en partie droite de f .

Cette restriction particulière ne change pas le pouvoir d'expression du formalisme : $L(LCFRS) = L(MCFG)$.

La 2-MCFG présentée plus haut pour le langage $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$, satisfaisant cette condition, est ainsi également un 2-LCFRS :

$$\begin{aligned} T &= \{0, 1\}, N = \{A, S\} \\ A &\rightarrow (0, 0) \mid (1, 1) \mid g_0[A] \mid g_1[A] \\ S &\rightarrow g_2[A] \\ g_0[(x, y)] &= (0x, 0y) \\ g_1[(x, y)] &= (1x, 1y) \\ g_2[(x, y)] &= xy \end{aligned}$$

3.3 Grammaires à concaténation d'intervalles

Une *grammaire à concaténation d'intervalles* (*range concatenation grammar* ou RCG, [Bou98]) est un ensemble de règles composées de *prédicats* sur des *intervalles* de chaînes de caractères.

Soient un ensemble de symboles non-terminaux $N = \{X, Y, Z, \dots\}$ et un ensemble de symboles terminaux $O = \{a, b, c, \dots\}$. Les règles d'une RCG sont de la forme :

$$\psi_0 \rightarrow \psi_1 \dots \psi_n$$

où les ψ_i sont des *prédicats* d'une arité p donnée :

$$\psi_i = A(\alpha_1, \dots, \alpha_p)$$

avec $\alpha_i \in (O \cup N)^*$. On distingue un *prédicat de départ*, S .

Les non-terminaux ne peuvent pas être utilisés plus d'une fois dans un

prédicat donné. On utilise également des prédicats *négatifs* : $\overline{A(X)}$ est vrai lorsque $A(X)$ est faux.

Les RCG opèrent sur des *intervalles* de la phrase.

Un *intervalle* est une sous-chaîne de mots du langage, et est défini formellement, étant donné une chaîne $w = a_1 \dots a_n$, par une paire (i, j) , $0 \leq i \leq j \leq n$.

(i, j) représente alors la sous-chaîne de w s'écrivant $a_{i+1} \dots a_j$. L'intervalle (i, j) est de taille $j - i$ et est vide quand $i = j$ ($(i, i) = \varepsilon$). Des intervalles consécutifs peuvent, s'ils sont cohérents, être concaténés, et $(i, j) \bullet (j, k) = (i, k)$.

Les arguments de prédicats dans une règle de RCG peuvent liés aux intervalles d'une chaîne w donnée, de façon cohérente quant à la concaténation : $A(aXb)$ peut par exemple porter sur n'importe chaîne commençant par a et finissant par b . Ainsi, $A(aX, bY) \rightarrow B(X, Y)$ peut être instancié par

$A((i, j), (k, l)) \rightarrow B((i + 1, j), (k + 1, l))$, si la chaîne w est telle que $a_{i+1} = a$ et $a_{k+1} = b$.

Une chaîne w est produite par une RCG G donnée si l'intervalle vide ε peut être dérivé d'une instanciation valide quelconque du prédicat de départ S , en utilisant les règles de G . Le *langage* de G est l'ensemble de ces chaînes.

La RCG suivante, de [Bou99], définit le langage $\{w^3, w \in \{a, b\}^*\}$:

$$\begin{aligned} S(XYZ) &\rightarrow A(X, Y, Z) \\ A(aX, aY, aZ) &\rightarrow A(X, Y, Z) \\ A(bX, bY, bZ) &\rightarrow A(X, Y, Z) \\ A(\varepsilon, \varepsilon, \varepsilon) &\rightarrow \varepsilon \end{aligned}$$

Une séquence de dérivation de *abaabaaba* étant :

$$S(abaabaaba) \rightarrow A(aba, aba, aba) \rightarrow A(ba, ba, ba) \rightarrow A(a, a, a) \rightarrow A(\varepsilon, \varepsilon, \varepsilon) \rightarrow \varepsilon$$

On peut également définir le langage $\{a^n b^n c^n d^n, n > 0\}$ par les règles suivantes :

$$\begin{aligned} S(XYZT) &\rightarrow A(X, Y, Z, T) \\ A(aX, bY, cZ, dT) &\rightarrow B(X, Y, Z, T) \end{aligned}$$

$$B(X, Y, Z, T) \rightarrow A(X, Y, Z, T)$$

$$B(\varepsilon, \varepsilon, \varepsilon, \varepsilon) \rightarrow \varepsilon$$

Dans ce cas, $aabbccdd$ serait reconnu suivant la suite de dérivations :

$$S(aabbccdd) \rightarrow A(aa, bb, cc, dd) \rightarrow B(a, b, c, d) \rightarrow A(a, b, c, d) \rightarrow B(\varepsilon, \varepsilon, \varepsilon, \varepsilon) \rightarrow \varepsilon$$

3.3.1 Grammaires à concaténation d'intervalles, simples et positives

Une RCG est dite *positive* quand elle ne contient aucun prédicat *négatif* (de la forme $\overline{A(X)}$). Cette propriété ne change pas la capacité générative des RCG générales :

$$L(PRCG) = L(RCG)$$

Une RCG G est dite *simple* quand elle satisfait l'ensemble des propriétés suivantes (analogues aux propriétés des LCFRS), soit quand G est :

Non-combinatoire : Une règle donnée $A_0(\dots) \rightarrow A_1(\dots) \dots A_k(\dots)$ n'utilise pas de symboles terminaux en partie droite, et ne définit les intervalles des $A_1 \dots A_k$ qu'avec les non-terminaux utilisés, en partie gauche, pour les intervalles de A_0 . (Autrement dit, on ne peut utiliser une règle de forme $A(X) \rightarrow B(aX)$.)

Linéaire : Une règle ne peut pas utiliser plusieurs fois de même symbole non-terminal en partie droite : on interdit $A(X) \rightarrow B(X)C(X)$.

Non-effaçante : Une règle doit utiliser chaque symbole non-terminal présent en partie gauche, une fois en partie droite : on interdit $A(X, Y) \rightarrow B(X)$.

Les *grammaires à concaténation d'intervalles simples et positives* (*simple positive range concatenation grammars* ou sPRCG) sont des RCG possédant ces deux propriétés. La classe de langage engendrée est égale à celle des MCFG, ce formalisme étant identique, aux considérations de présentation près, aux LCFRS. (Ces résultats sont détaillés dans [Bou98] et présentés ici section 4.1.2.)

3.4 Grammaires Minimalistes

Les *grammaires minimalistes* (*minimalist grammars* ou MG), définies dans [Sta97b] par Edward Stabler, sont une adaptation formelle du *programme minimaliste* de Noam Chomsky. Ce *programme minimaliste* représente, de façon épurée et réduite à l'essentiel, les mécanismes qui semblent être communs à l'ensemble des langues humaines, et feraient donc partie d'une hypothétique *grammaire universelle*, supposée innée dans l'être humain qui, paramétrée, pourrait se dériver dans chacune des langues. Quelle que soit la vraisemblance de ces hypothèses, le programme minimaliste, par construction, peut exprimer tous les phénomènes étudiés pendant la période de son élaboration.

Son principe est fondé sur la notion de *mouvement*, et permet de rendre facilement compte du changement de forme de l'énoncé³. Par conséquent, Stabler a décidé d'organiser les grammaires minimalistes autour d'une structure arborescente spécialisée, permettant de tenir facilement compte des opérations à effectuer.

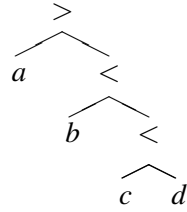
3.4.1 Structure d'arbres ordonnés avec projection

Une définition formelle et détaillée de cette structure est donnée dans [Sta97b] ou [Amb03]. De façon informelle, les arbres utilisés pour les grammaires minimalistes sont finis, ordonnés, en général binaires, et possèdent une relation de *projection*, qui donne un ordre entre les sous-arbres venant se rajouter à l'ordre gauche-droite classique. Seules les feuilles sont étiquetées, les nœuds internes pouvant servir à représenter la relation de projection : le fait qu'une feuille n se projette sur une autre feuille m peut

être noté $n < m$ et représenté par l'arbre $\begin{array}{c} < \\ \wedge \\ n \quad m \end{array}$.

Ainsi, en considérant l'arbre suivant :

³Par exemple, la forme interrogative d'une phrase affirmative en français met en jeu le mouvement du sujet ou de l'objet autour du verbe.



Ici, c se projette sur d , par exemple. Pour des raisons de commodité, on définit également la *tête* d'un arbre (ou sous-arbre) comme la feuille qui maximise la relation de projection pour l'arbre (ou le sous-arbre) : ici, c est

la tête de $\begin{array}{c} < \\ \wedge \\ c \quad d \end{array}$, et b est la tête de l'arbre.

La tête d'un sous-arbre enraciné en t peut être facilement obtenue par un parcours en profondeur dans l'ordre de projection, jusqu'à obtention d'une feuille.

On définit aussi la *projection maximale* d'une feuille comme le plus petit nœud enracinant un sous-arbre dont cette feuille soit la tête : ici, a est sa propre projection maximale, et la racine est la projection maximale de b .

3.4.2 Traits et lexiques

Les grammaires minimalistes sont entièrement définies par des *entrées lexicales*, elles-mêmes composées de *traits (features)* qui représentent certaines caractéristiques. Ces traits peuvent être des :

Traits phonologiques : modélisant le mot, la partie de mot ou le groupe de mots de façon phonétique (représentant le son produit) ou écrite (représentant la lettre), suivant l'application voulue. Ces traits sont ceux qui sont directement visibles dans une phrase donnée, qu'elle soit produite par la grammaire ou destinée à l'analyse.

Traits sémantiques : modélisant le sens du trait phonétique correspondant. Pour simplifier, dans le cas, comme ici, d'un formalisme non destiné à une analyse sémantique, logique ou pragmatique, on pourra ignorer ces traits, mais leur inclusion est prévue dans la spécification des grammaires.

Traits syntaxiques : modélisant la structure qui sera construite par dérivation, et qui doit approcher la construction syntaxique de la phrase.

Pour les besoins du formalisme, les ensembles suivants de traits sont définis :

Traits non syntaxiques : L'ensemble V des traits *non-syntaxiques* (soit des traits phonétiques et sémantiques).

Catégories de base : L'ensemble B des *catégories de base*, sous-ensemble des traits syntaxiques. Cet ensemble regroupe les catégories syntaxiques de la phrase, soit en français v pour un verbe, n pour un nom, etc. On distingue au moins un *trait acceptant* $c \in B^4$, qui jouera un rôle analogue aux symboles initiaux S des grammaires génératives.

Sélecteurs : L'ensemble S des *sélecteurs* (*selectors*), notés $= D$ avec $D \in B$. Ces traits correspondent à une demande (ou attente) d'une catégorie de base à un endroit donné.

Assignateurs : L'ensemble $+L$ des *assignateurs* (*licensors*), $+f$. Ces traits correspondent à une *propriété*, pouvant donner lieu à un mouvement. Dans les langues, ces traits sont, par exemple, utilisés pour la représentation des *cas* (nominatif, ablatif...), correspondant aux fonctions grammaticales, ou encore pour noter le caractère interrogatif d'une structure...

Assignés : L'ensemble $-L$ des *assignés* (*licensees*), $-f$. Symétriquement aux assignateurs, ces traits indiquent une demande de propriété. Un assigné $-f$ correspond à l'attente d'une propriété exprimée par l'assignateur $+f$; dans le cas des langues, le cas est requis par la structure à laquelle appartient $-f$ et apporté par une expression possédant le trait $+f$.

Une *entrée*, ou *expression*, d'une grammaire minimaliste donnée peut être de deux types : *lexicale*, donc définie dans le lexique de la grammaire, notée par $\{::\}$, ou *dérivée*, donc construite par dérivation des entrées lexicales, notée par $\{:\}$.

Le lexique de la grammaire est alors un ensemble d'entrées de la forme :

$$V^* \times \{::\} \times (B \cup S \cup +L \cup -L)^*$$

⁴ c ou C correspond à *complementizer*, pour noter une structure syntaxique, éventuellement vide, qui pourrait s'ajouter à une phrase complète. Ce complément optionnel ne change pas le caractère de la phrase, qui fait partie du langage dérivé. Ce peut être le mot vide, le *Que ... !* d'exclamation français, le *Say*, anglais, le *Nee* japonais, ...

Il est d'usage de se limiter à des entrées formatées comme suit :

$$V^* \times \{::\} \times S^* \times (+L)^? \times S^* \times B \times (-L)^*$$

Cependant, de nombreuses variantes existent (certaines grammaires utilisent notamment des entrées à plus d'un assignateur). Ces variations de formes des entrées ne changent pas, en elles-mêmes, le formalisme.

3.4.3 Grammaires Minimalistes classiques

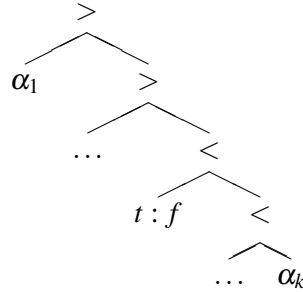
Une grammaire minimaliste est entièrement définie par son lexique. Les entrées du lexique sont dérivées suivant les opérations de *fusion* (*merge*) et de *mouvement* (*move*).

Une expression, ne contenant plus que le (ou l'un des) *traits acceptants* de la grammaire en tant que trait syntaxique, est terminale, et ses traits non-syntaxiques appartiennent alors au langage de la grammaire.

Les étapes de la dérivation peuvent être visualisées sur un arbre fini, ordonné, avec projection, qui donne la structure syntaxique d'une phrase analysée ou produite.

Considérons $s, t \in V^*$ des chaînes de traits non-syntaxiques (typiquement, des mots), $f \in B$ une catégorie de base, $= f \in S$ le sélecteur correspondant, $\gamma \in (B \cup S \cup +L \cup -L)^*$ une chaîne (éventuellement vide) de traits syntaxiques, $\delta \in (B \cup S \cup +L \cup -L)^+$ une chaîne non-vide de traits syntaxiques, $\cdot \in \{:, ::\}$ indiquant le type de l'expression ($\{::\}$ pour une entrée lexicale, $\{:\}$ pour une expression dérivée, et des α_i et β_i un certain nombre (éventuellement nul) d'autres expressions simples.

Ainsi, l'expression $t : f, \alpha_1, \dots, \alpha_k$, dont la tête est l'expression simple $t : f$, et les α_i représentent d'autres expressions simples, qui n'interviendront pas immédiatement dans une dérivation, représente, par exemple, l'arbre suivant :



Les opérations de dérivation sont alors définies comme suit :

Fusion : La fusion (*merge*) de deux expressions correspond à la réalisation d'un besoin de sélection, c'est-à-dire l'intégration, par une expression comportant un sélecteur $= D$, d'une expression comportant le trait de base D . Cette opération est différente suivant les situations :

Sélection d'une expression simple par une entrée lexicale : L'entrée lexicale $s ::= f\gamma$, est combinée avec l'expression $t \cdot f, \alpha_1, \dots, \alpha_k$ (la tête, soit la première partie de l'expression, ne comporte ici qu'un trait de base). On forme alors l'expression dérivée suivante :

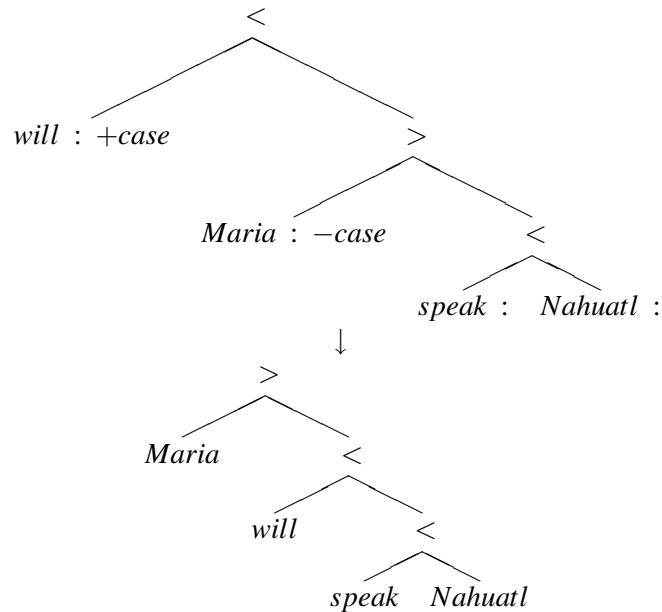
$$\frac{s ::= f\gamma \quad t \cdot f, \alpha_1, \dots, \alpha_k}{st : \gamma, \alpha_1, \dots, \alpha_k}$$

Sélection d'une expression simple par une expression dérivée :

L'expression dérivée $s := f\gamma, \alpha_1, \dots, \alpha_k$, est combinée avec l'expression $t \cdot f, \beta_1, \dots, \beta_l$ (ici encore, la tête ne comporte qu'un trait de base). On forme alors l'expression dérivée suivante (de manière analogue à la précédente, mais les traits non-syntaxiques sont présentés dans l'ordre inverse) :

$$\frac{s := f\gamma\alpha_1, \dots, \alpha_k \quad t \cdot f, \beta_1, \dots, \beta_l}{ts : \gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l}$$

Sélection d'une expression composite : L'expression (lexicale ou dérivée) $s \cdot = f\gamma, \alpha_1, \dots, \alpha_k$ est combinée avec l'expression $t \cdot f\delta, \beta_1, \dots, \beta_l$ (ici, par contre, la tête comporte un ou des traits qui causeront une fusion supplémentaire, ou un déplacement). On forme alors l'expression suivante :



3.4.4 Grammaires Minimalistes augmentées

La définition des grammaires minimalistes, donnée plus haut, ne correspond pas exactement à celle de [Sta97b]. En effet, l'article original fait appel à des notions supplémentaires, notamment la notion de traits *forts* $+X$ et *faibles* $-x$, permettant d'effectuer un mouvement portant sur l'expression entière, comme défini précédemment, ou un mouvement laissant les traits phonétiques à l'emplacement précédent, dit *mouvement caché* ou *covert phrasal move*, ainsi qu'une opération de *mouvement de tête*, ou *head movement*.

Cependant, [Mic01a] prouve que ces notions ne changent pas la classe de langages exprimés. D'autres fonctionnalités, comme l'adjonction, ont été postérieurement ajoutées aux grammaires minimalistes, formant autant de variantes appelées ici collectivement *grammaires minimalistes augmentées* ou hMG.

Nous ne détaillerons pas, ici, les modifications à apporter pour ces fonctionnalités, mais il s'agit, en général, d'ajouter les cas adéquats aux opérations concernées de fusion et de mouvement. Ces variantes, bien qu'elles ne changent pas la classe de langages du formalisme, permettent d'exprimer plus directement certains phénomènes linguistiques.

Il est aussi possible d'étendre les grammaires minimalistes pour obtenir

des formalismes dotés d'un pouvoir d'expression plus important : [MK05] propose ainsi deux variantes de type-0 (donc bien au-delà des langages légèrement contextuels...).

Chapitre 4

Équivalences et conversions des formalismes

Sans langage commun, les affaires ne peuvent être conclues.

Kong Fu Zi

4.1 Variantes des PMCFG, MCFG et RCG

4.1.1 PMCFG et RCG

Dans [Lju05], Peter Ljunglöf étend les PMCFG avec une opération d'*intersection*. La classe de langages induite par la clôture de l'intersection sur l'ensemble des PMCFG est identique à celle des RCG, donc à *PTIME*.

4.1.2 LCFRS et sPRCG

Il est remarqué dans [Bou98] que les LCFRS et les sPRCG sont fortement (et faiblement) équivalents. En fait, ces deux formalismes, bien que définis à partir de motivations très différentes, sont identiques à l'écriture près. Une règle de LCFRS sous la forme :

$$\begin{aligned} A &\rightarrow g(B_1, B_2, \dots, B_m) \\ g(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m) &= \vec{t} \end{aligned}$$

Où les \vec{x}_i et \vec{t} sont des variables représentant des tuples de chaînes (notamment, \vec{t} est un réarrangement des composantes des \vec{x}_i), est trivialement équivalente à une règle de sPRCG de la forme :

$$A(\vec{t}) \rightarrow B_1(\vec{x}_1), B_2(\vec{x}_2), \dots, B_m(\vec{x}_m)$$

Les prédicats jouent ici le rôle des non-terminaux du LCFRS.

4.1.3 Forme normale des MCFG

[SMFK91] introduit la notion de *forme normale* pour les MCFG. Il y est démontré que, pour toute MCFG G , il existe une MCFG G' faiblement équivalente, qui vérifie :

- Toute variable d'une fonction de G' est présente exactement une fois en partie droite (autrement dit, G' est un LCFRS).
- G' est propre : aucune de ses règles ne fait appel à des chaînes constantes, mises à part les règles terminales.

De plus, [Mic01a] remarque qu'on peut considérer, sans perte de généralité, que les fonctions des MCFG prennent au plus deux arguments (il suffit de décomposer en plusieurs règles successives quand ce n'est pas le cas).

Ainsi, [Gui04] définit et utilise une *forme normale binaire* des MCFG, qui regroupe les conditions précédentes. Pour toute MCFG, il existe une MCFG faiblement équivalente en forme normale binaire. On peut noter les règles d'une MCFG en forme normale binaire sous cette forme :

$$A \rightarrow BC[0, 1; 1, 0][0, 0][1, 1]$$

A supposer que A soit d'arité 3 et que B et C soient d'arité 2, les coordonnées $[i, j]$ représentent les variables de B et C à extraire, et cette formulation est équivalente à :

$$A \rightarrow g[B, C]$$

$$g[(a, b), (c, d)] = (bc, a, d)$$

4.2 Équivalence MG-MCFG et analyse des grammairistes minimalistes

[Mic01a] prouve que, pour toute grammaire minimaliste, il existe une MCFG qui engendre le même langage. [Mic01d] prouve que, pour tout LC-

FRS, il existe une grammaire minimaliste engendrant le même langage, et donc que ces trois formalismes partagent la même classe de langages. De plus, [Mic01a] construit formellement la MCFG correspondant à une grammaire minimaliste donnée. Cette conversion est détaillée et implémentée de façon efficace dans [Gui04].

4.2.1 Procédure de conversion des grammaires minimalistes en MCFG

La conversion d'une grammaire minimaliste G_{MG} en MCFG G consiste à représenter, à l'aide des non-terminaux de la MCFG, l'ensemble de la structure de la grammaire minimaliste, du point de vue des opérations de fusion et de mouvement possibles.

[Mic01a] fait remarquer que, étant donné le lexique fini d'une grammaire minimaliste, l'ensemble des expressions dérivables à partir du lexique est lui aussi fini.

En effet, chaque opération de fusion ou de mouvement efface deux des symboles des expressions utilisées (sélecteur et catégorie, ou assignateur et assigné). De plus, les expressions ne peuvent être constituées que des suffixes des entrées lexicales. En faisant correspondre à un symbole non-terminal de MCFG toutes les expressions utilisées (qu'elles soient lexicales ou dérivées), on modélise les opérations de dérivation permettant la constitution de ces expressions par des règles portant sur les symboles qui leur sont associés : un symbole dans le cas d'un mouvement, deux dans le cas d'une fusion.

Ainsi, la MCFG construite sera en forme normale binaire.

[Gui04] explicite cette procédure. Nous utilisons les notations définies précédemment pour les règles des grammaires minimalistes, notamment : $\cdot \in \{:, ::\}$ représente le type de l'expression ($\{::\}$ pour une entrée lexicale, $\{:\}$ pour une expression dérivée).

1. Par convention, à toute expression e (constituée de traits syntaxiques uniquement) on associe un symbole $\sigma(e)$. Ces symboles seront notés t_0, t_1, \dots
2. Pour toute entrée lexicale $s :: \delta$, on ajoute à la MCFG la règle terminale suivante : $\sigma(\delta) \rightarrow s$.
3. Pour le symbole acceptant C , on ajoute la règle initiale : $S \rightarrow \sigma(C)[0, 0]$ – ou, en notation usuelle :

$$\begin{array}{l} S \rightarrow f[\sigma(C)] \\ f[x] = x \end{array}$$

4. Lorsque l'on utilise la règle suivante en MG :

$$\frac{s ::= f\gamma \quad t \cdot f, \alpha_1, \dots, \alpha_k}{st : \gamma, \alpha_1, \dots, \alpha_k}$$

On ajoute à la MCFG le symbole $\sigma(\gamma, \alpha_1, \dots, \alpha_k)$, et la règle :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_k) \rightarrow \sigma(= f\gamma)\sigma(f, \alpha_1, \dots, \alpha_k)[0, 0; 1, 0] [1, 1] \dots [1, k]$$

Qui s'écrit également :

$$\begin{array}{l} \sigma(\gamma, \alpha_1, \dots, \alpha_k) \rightarrow g[\sigma(= f\gamma), \sigma(f, \alpha_1, \dots, \alpha_k)] \\ g[(x_0), (y_0, y_1, \dots, y_k)] = (x_0 y_0, y_1, \dots, y_k) \end{array}$$

5. Lorsque l'on utilise la règle suivante en MG :

$$\frac{s := f\gamma\alpha_1, \dots, \alpha_k \quad t \cdot f, \beta_1, \dots, \beta_l}{ts : \gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l}$$

On ajoute à la MCFG le symbole $\sigma(\gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l)$, et la règle :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l) \rightarrow \sigma(= f\gamma, \alpha_1, \dots, \alpha_k)\sigma(f, \beta_1, \dots, \beta_l)[1, 0; 0, 0] [0, 1] \dots [0, k] [1, 1] \dots [1, l]$$

Qui s'écrit également :

$$\begin{array}{l} \sigma(\gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l) \rightarrow g[\sigma(= f\gamma, \alpha_1, \dots, \alpha_k), \sigma(f, \beta_1, \dots, \beta_l)] \\ g[(x_0, x_1, \dots, x_k), (y_0, y_1, \dots, y_l)] = (y_0 x_0, x_1, \dots, x_k, y_1, \dots, y_l) \end{array}$$

6. Lorsque l'on utilise la règle suivante en MG :

$$\frac{s \cdot = f\gamma\alpha_1, \dots, \alpha_k \quad t \cdot f\delta, \beta_1, \dots, \beta_l}{s : \gamma, \alpha_1, \dots, \alpha_k, t : \delta, \beta_1, \dots, \beta_l}$$

On ajoute à la MCFG le symbole $\sigma(\gamma, \alpha_1, \dots, \alpha_k, \delta, \beta_1, \dots, \beta_l)$, et la règle :

$$\begin{array}{l} \sigma(\gamma, \alpha_1, \dots, \alpha_k, \delta, \beta_1, \dots, \beta_l) \rightarrow \\ \sigma(= f\gamma, \alpha_1, \dots, \alpha_k) \sigma(f\delta, \beta_1, \dots, \beta_l)[0, 0] \dots [0, k] [1, 0] \dots [1, l] \end{array}$$

Qui s'écrit également :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_k, \delta, \beta_1, \dots, \beta_l) \rightarrow g[\sigma(=f\gamma, \alpha_1, \dots, \alpha_k), \sigma(f\delta, \beta_1, \dots, \beta_l)]$$

$$g[(x_0, x_1, \dots, x_k), (y_0, y_1, \dots, y_l)] = (x_0, x_1, \dots, x_k, y_0, y_1, \dots, y_l)$$

7. Lorsque l'on utilise la règle suivante en MG :

$$\frac{s : +f\gamma\alpha_1, \dots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \dots, \alpha_k}{tS : \gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k}$$

On ajoute à la MCFG le symbole $\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k)$, et la règle :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k) \rightarrow \sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f, \alpha_{i+1}, \dots, \alpha_k)$$

$$[0, i; 0, 0] [0, 1] \dots [0, i-1] [0, i+1] \dots [0, k]$$

Qui s'écrit également :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k) \rightarrow g[\sigma(+f\gamma, \alpha_1, \dots, \alpha_k)]$$

$$g[(x_0, x_1, \dots, x_k)] = (x_i x_0, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$$

8. Lorsque l'on utilise la règle suivante en MG :

$$\frac{s : +f\gamma\alpha_1, \dots, \alpha_{i-1}, t : -f\delta, \alpha_{i+1}, \dots, \alpha_k}{s : \gamma, \alpha_1, \dots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \dots, \alpha_k}$$

On ajoute à la MCFG le symbole $\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k)$, et la règle :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k) \rightarrow$$

$$\sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, +f\delta, \alpha_{i+1}, \dots, \alpha_k)$$

$$[0, 0] [0, 1] \dots [0, k]$$

Qui s'écrit également :

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k) \rightarrow g[\sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, +f\delta, \alpha_{i+1}, \dots, \alpha_k)]$$

$$g[(x_0, x_1, \dots, x_k)] = (x_0, x_1, \dots, x_k)$$

La MCFG résultante rend donc exactement compte de toutes les possibilités de combinaison des entrées lexicales intrinsèques à la MG de départ. Par construction, les étapes de dérivation lors de l'analyse ou de la production d'une phrase sont exactement équivalentes. Cependant, la MCFG ne permet pas d'obtenir directement l'arbre structurel d'une phrase, au contraire de la MG ; pour retrouver cette structure à fin d'analyses supplémentaires, il est nécessaire de conserver l'historique des règles utilisées. En effet, la dérivation par la MCFG ne donne que la *chaîne* produite, soit le dernier niveau d'un arbre de grammaire minimaliste (qui, lui, a conservé la trace des mouvements effectués, par exemple).

D'autre part, [Gui04] propose également des procédures portant sur plusieurs variantes des grammaires minimalistes (hMG), faisant correspondre à chaque dérivation une règle de MCFG, de façon semblable à la conversion présentée ici en détail.

4.2.2 Exemple de conversion

Nous utiliserons ici un exemple de grammaire engendrant les langages à compteurs, de type $\{a_1^n a_2^n \dots a_k^n\}$, dont les implémentations par des grammaires minimalistes ont été réalisées par Maxime Amblard (en utilisant la forme d'entrée autorisant plusieurs assignateurs).

La procédure est identique pour toutes les grammaires de ce type, nous détaillerons le cas de $\{a^n b^n\}$, testé sous cette forme avec l'implémentation de Matthieu Guillaumin. Nous présenterons ici les règles de la MCFG résultante avec la notation binaire normale présentée auparavant.

La grammaire minimaliste G_{MG} , dont les catégories sont $\{D, V, C\}$ et le symbole acceptant est C , générant le langage $\{a^n b^n\}$, a pour lexique :

$$\begin{aligned}
 b &:: D - D \\
 b &:: = V + D D - D \\
 a &:: = D V - V \\
 a &:: = D + V V - V \\
 &:: C \\
 &:: = V + D + V C
 \end{aligned}$$

Pour construire la MCFG G_{MCFG} équivalente, on commence par ajou-

ter les symboles non-terminaux t_0 à t_5 , correspondant aux expressions suivantes (directement extraites des entrées lexicales) :

$$\begin{aligned}
 t_0 &= \sigma(:: D - D) \\
 t_1 &= \sigma(:: = V + D D - D) \\
 t_2 &= \sigma(:: = D V - V) \\
 t_3 &= \sigma(:: = D + V V - V) \\
 t_4 &= \sigma(:: C) \\
 t_5 &= \sigma(:: = V + D + V C)
 \end{aligned}$$

Le lexique donne aussi, directement, les règles terminales suivantes de G_{MCFG} :

$$\begin{aligned}
 t_0 &\rightarrow b \\
 t_1 &\rightarrow b \\
 t_2 &\rightarrow a \\
 t_3 &\rightarrow a \\
 \hline
 t_4 &\rightarrow \varepsilon \\
 t_5 &\rightarrow \varepsilon
 \end{aligned}$$

Pour terminer l'initialisation de l'algorithme, on ajoute à G_{MCFG} le symbole initial S , et la règle :

$$S \rightarrow t_4[0, 0]$$

Ensuite, on applique la clôture des dérivations possibles à partir des entrées lexicales, en ajoutant à chaque étape les symboles et règles adéquats :

- Les symboles t_0 et t_3 , modélisant les expressions $D - D$ et $= D + V V - V$, peuvent être combinés pour modéliser une fusion entre $= D$ et D , formant un hypothétique symbole t_6 représentant $+V V - V$, $-D$. Cependant, cette expression composite est insoluble : aucune dérivation possible ne peut résoudre le premier symbole de sa tête, $+V$ (en particulier, le trait $-V$ n'est présent dans aucune des autres composantes de cette expression, ce qui interdit tout mouvement). Dans ce cas, l'algorithme abandonne cette branche de dérivation et n'ajoute ni symbole ni règle à G_{MCFG} .

- Les symboles t_0 et t_2 , modélisant les expressions $D - D$ et $= D V - V$, peuvent être combinés pour modéliser une fusion entre $= D$ et D , formant un symbole t_7 représentant l'expression $V - V, -D$. Ici, contrairement au cas précédent, une règle peut être appliquée au symbole résultant, utilisant une fusion avec une autre expression, dont le trait serait $=D$ (expression qui est déjà présente dans le lexique). On ajoute ainsi à G_{MCFG} le symbole t_7 et la règle :

$$t_7 \rightarrow t_2 t_0 [0, 0] [1, 0]$$

- Les symboles t_5 et t_7 , modélisant les expressions $= V + D + V C$ et $V - V, -D$, peuvent être combinés pour modéliser une fusion $= V / V$, formant un symbole t_8 représentant $+D + V C, -V, -D$. Là aussi, une règle de mouvement peut être appliquée à cette nouvelle expression, portant sur deux de ses comosantes : sa tête commençant par $+D$ et une autre, commençant par $-D$. On ajoute ainsi à G_{MCFG} le symbole t_8 et la règle :

$$t_8 \rightarrow t_5 t_7 [0, 0] [1, 0] [1, 1]$$

- Les symboles t_1 et t_7 , modélisant les expressions $= V + D D - D$ et $V - V, -D$, peuvent être combinés pour modéliser une fusion $= V / V$, formant un symbole t_9 représentant $+D D - D, -V, -D$. On ajoute ainsi à G_{MCFG} le symbole t_9 et la règle :

$$t_9 \rightarrow t_1 t_7 [0, 0] [1, 0] [1, 1]$$

- Le symbole t_9 , modélisant l'expression $+D D - D, -V, -D$, peut être modifié pour modéliser un mouvement $+D / -D$, formant un symbole t_{10} représentant $D - D, -V$. On ajoute ainsi à G_{MCFG} le symbole t_{10} et la règle :

$$t_{10} \rightarrow t_9 [0, 2; 0, 0] [0, 1]$$

- Le symbole t_8 , modélisant l'expression $+D + V C, -V, -D$, peut être modifié pour modéliser un mouvement $+D / -D$, formant un symbole t_{11} représentant $+V C, -V$. On ajoute ainsi à G_{MCFG} le symbole t_{11} et la règle :

$$t_{11} \rightarrow t_8 [0, 2; 0, 0] [0, 1]$$

- Le symbole t_{11} , modélisant l'expression $+V C, -V$, peut être modifié pour modéliser un mouvement $+V / -V$, formant un symbole t_{12} représentant : C . On ajoute ainsi à G_{MCFG} le symbole t_{12} et la règle :

$$t_{12} \rightarrow t_{11} [0, 1; 0, 0]$$

- Les symboles t_3 et t_{10} , modélisant les expressions $= D + V V - V$ et $D - D, -V$, peuvent être combinés pour modéliser une fusion $= D/D$, formant un symbole t_{13} représentant $+V V - V, -D, -V$. On ajoute ainsi à G_{MCFG} le symbole t_{13} et la règle :

$$t_{13} \rightarrow t_3 t_{10} [0, 0] [1, 0] [1, 1]$$

- Le symbole t_{13} , modélisant l'expression $+V V - V, -D, -V$, peut être modifié pour modéliser un mouvement $+V / -V$. L'expression formée par cette dérivation est $V - V, -D$, qui correspond au symbole t_7 . On se contente donc d'ajouter à G_{MCFG} la règle :

$$t_7 \rightarrow t_{13} [0, 2; 0, 0] [0, 1]$$

Il n'y a pas d'autre dérivation possible de ces expressions. Pour terminer, on ajoute la règle suivante, correspondant au symbole t_{12} , créé dans les étapes précédentes, qui correspond au symbole acceptant C :

$$S \rightarrow t_{12} [0, 0]$$

On obtient, au final, une MCFG G_{MCFG} comportant les symboles non-terminaux suivants :

$$\begin{aligned}
& S \\
& t_0 = \sigma(:: D - D) \\
& t_1 = \sigma(:: = V + D D - D) \\
& t_2 = \sigma(:: = D V - V) \\
& t_3 = \sigma(:: = D + V V - V) \\
& t_4 = \sigma(:: C) \\
& t_5 = \sigma(:: = V + D + V C) \\
& t_6 \text{ (non utilisé)} \\
& t_7 = \sigma(: V - V, -D) \\
& t_8 = \sigma(: +D + V C, -V, -D) \\
& t_9 = \sigma(: +D D - D, -V, -D) \\
& t_{10} = \sigma(: D - D, -V) \\
& t_{11} = \sigma(: +V C, -V) \\
& t_{12} = \sigma(: C) \\
& t_{13} = \sigma(: +V V - V, -D, -V)
\end{aligned}$$

Les règles de G_{MCFG} sont les suivantes :

$$\begin{aligned}
S &\rightarrow t_4[0,0] \\
t_4 &\rightarrow \varepsilon \\
S &\rightarrow t_{12}[0,0] \\
t_{12} &\rightarrow t_{11}[0,1;0,0] \\
t_{11} &\rightarrow t_8[0,2;0,0] [0,1] \\
t_8 &\rightarrow t_5t_7[0,0] [1,0] [1,1] \\
t_5 &\rightarrow \varepsilon \\
t_7 &\rightarrow t_{13}[0,2;0,0] [0,1] \\
t_{13} &\rightarrow t_3t_{10}[0,0] [1,0] [1,1] \\
t_3 &\rightarrow a \\
t_{10} &\rightarrow t_9[0,2;0,0] [0,1] \\
t_9 &\rightarrow t_1t_7[0,0] [1,0] [1,1] \\
t_1 &\rightarrow b \\
t_7 &\rightarrow t_2t_0[0,0] [1,0] \\
t_2 &\rightarrow a \\
t_0 &\rightarrow b
\end{aligned}$$

4.2.3 Complexité de la conversion

Edward Stabler a remarqué que, en utilisant la procédure de conversion telle qu'elle est implémentée ici, on pouvait avoir un problème de complexité : pour certaines grammaires minimalistes, la MCFG obtenue par conversion pourrait être exponentiellement plus grande que la grammaire de départ. Ce serait le cas, notamment, des grammaires données par Jens Michaelis pour les langages à compteurs ($\{a_1^n a_2^n \dots a_k^n\}$ pour k croissant), dont la conversion en MCFG croît très rapidement, alors qu'il existe des MCFG plus simples décrivant les mêmes langages.

Notre hypothèse est que, la conversion ne portant que sur la structure syntaxique intrinsèque donnée par la grammaire minimaliste, la taille de la MCFG obtenue, et donc la complexité de la conversion, ne dépend que de la manière d'écrire la grammaire minimaliste au départ, et que le fait que la MCFG soit exponentielle en taille par rapport à la grammaire de départ signifie simplement que celle-ci n'est pas efficace – notamment, que la conversion ne peut pas être de complexité significativement supérieure à la résolution du problème d'analyse d'une phrase quelconque à l'aide de la grammaire de départ.

Cette hypothèse a été confortée par le fait que, pour les langages à compteurs utilisés précédemment, la conversion réalisée sur des grammaires minimalistes cette fois décrites par Maxime Amblard donne de très

bons résultats : pour une grammaire de départ de taille $2n + O(1)$, la MCFG obtenue est de taille $3n + O(1)$.

Un problème de cette approche est qu'il n'est pas immédiatement apparent qu'une grammaire minimaliste donnée soit "meilleure" qu'une autre, à langage fixé. Nous ne disposons d'aucun algorithme permettant, en particulier, de démontrer qu'une grammaire soit minimale pour un langage...

Nous avons ensuite tenté de borner la complexité en taille de la MCFG obtenue. Il apparaît qu'une grammaire minimaliste peut effectivement générer une MCFG de très grande taille, sous certaines conditions ; le paramètre le plus important est la longueur maximale d'une entrée lexicale, et notamment les nombres maximaux de sélecteurs et d'assignateurs dans une entrée. Nous pouvons donner deux arguments simples à cet effet, en considérant une grammaire aux possibilités de dérivations arbitrairement nombreuses.

Analyse par étapes de l'algorithme de conversion

Tout d'abord, considérons une grammaire minimaliste composée de n entrées lexicales, comportant chacune au plus s sélecteurs et m assignateurs.

On applique l'algorithme de conversion par *étapes*, cherchant, à chaque itération, à appliquer toutes les dérivations possibles aux expressions déjà obtenues. Notons n_i le nombre maximal d'expressions produites à l'étape i .

Nous avons $n_0 = n$, la première étape n'intégrant que les entrées lexicales, et $n_1 = n^2$, les seules dérivations possibles au début étant les combinaisons (par fusion) d'entrées lexicales. Ensuite, à une étape i quelconque, les n_{i-1} expressions produites à l'étape précédente peuvent fusionner chacune avec toutes les autres entrées produites auparavant. Le nombre maximal d'expressions produites par fusion à l'étape i est donc de $g_i = n_{i-1} \sum_{k=0}^{i-2} (n_k)$.

A cette même étape i , tous les éléments produits à l'étape précédente peuvent également subir un mouvement. Le nombre maximal de mouvements est donc $v_i = n_{i-1}$. On a $n_i = g_i + v_i$, soit $n_i = n_{i-1} (1 + \sum_{k=0}^{i-2} (n_k))$. D'autre part, chaque entrée lexicale ne peut, au plus, subir que s fusions et m mouvements, le nombre d'étapes est donc borné par $s + m$: il ne peut y avoir, au plus, que $n + n^2 + \sum_{i=2}^{s+m} (n_i)$ expressions générées au total.

On peut réduire cette complexité à $O(n^p)$, où p est le plus fort exposant de cette somme, soit celui de n_{s+m} . Examinons alors E_i , le plus fort exposant

de n_i : on a $E_0 = 1$ (car $n_0 = n$), $E_1 = 2$, et $E_i = E_{i-1} + E_{i-2}$ (car le plus fort exposant de $n_i = n_{i-1}(1 + \sum_{k=0}^{i-2} n_k)$ est celui de $(n_{i-1})(n_{i-2})$), et on peut donc remarquer que $E_i = F_{i+2}$, où F est la suite de Fibonacci usuelle.

La complexité, en temps, de la conversion, est donc (au pire) de $O(n^{F_{s+m+2}})$, ce qui est très fortement croissant avec $s + m$ (de l'ordre de $O(n^{4^{s+m}})$).

Analyse par quantité maximale d'expressions dérivables

On peut également considérer une grammaire composée de n entrées lexicales, chacune de longueur au plus l et comportant au plus s sélecteurs.

On peut alors observer que chaque expression dérivée comporte au plus 2^s composantes. En effet, seule l'opération de fusion permet d'adjoindre de nouvelles composantes à une expression, en nombre égal à celle de l'expression adjointe. La tête de l'expression étant formée à partir d'une entrée lexicale, celle-ci ne peut subir qu'au plus s fusions, au cours desquelles le nombre de composantes est au maximum doublé.

De plus, chaque dérivation consommant le premier symbole d'une ou plusieurs des composantes, celles-ci ne peuvent être formées que des suffixes des entrées lexicales. Or, il n'y a au plus que l suffixes pour chaque entrée, et donc nl suffixes au total. Il ne pourrait donc y avoir, au plus, que $(nl)^{2^s}$ expressions dérivables à partir du lexique.

La complexité en taille de la MCFG obtenue est donc inférieure à $O((nl)^{2^s})$, résultat (en général) meilleur que le précédent. Cependant, elle ne rend pas compte de toutes les étapes de l'algorithme, la quantité de temps et de mémoire utilisés pour la conversion est donc supérieure.

4.2.4 Analyse des grammaires minimalistes

Depuis la formalisation des grammaires minimalistes, plusieurs analyseurs (*parsers*) pour ces formalismes ont été implémentés, à commencer par celui de Harkema, détaillé dans [Har01]. Ces programmes sont, en général, des variantes de l'algorithme de Cocke-Younger-Kasami pour les grammaires algébriques (présenté, par exemple, dans [Kas65]).

Un des aspects pratiques de la recherche de l'équivalence des grammaires minimalistes est de comparer la performance de ces analyseurs avec une procédure consistant à convertir les grammaires minimalistes dans un autre formalisme, puis d'appliquer au résultat un analyseur déjà connu et

performant pour le formalisme d'arrivée. Il existe ainsi un analyseur pour les MCFG proposé par Daniel Albro dans [Alb00], qui n'est pas entièrement correct car il ne donne les résultats attendus que pour les MCFG en forme normale binaire – ce qui convient parfaitement au résultat d'une conversion de grammaire minimaliste. Cet algorithme est une variante de l'algorithme de Earley (présenté dans [Ear70]). La complexité de l'analyse d'une phrase de n mots avec une MCFG de c catégories (nombre maximal de variables utilisées dans une règle quelconques) est de $O(n^{4c+3} \log_2 n)$.

Dans [Gui04], Guillaumin teste sur quelques exemples de phrases, pour une grammaire minimaliste, les performances des différents algorithmes à sa disposition et de la procédure de conversion-analyse de la MCFG résultante. Les résultats sont sans appel, l'analyse après conversion étant plus rapide que le meilleur algorithme d'analyse des grammaires minimalistes de un (0,2 seconde contre 5 secondes environ) à quatre (0,6 seconde contre 43 minutes environ) ordres de magnitude. Les résultats expérimentaux semblent donc valider cette méthode, même si les tests effectués étaient très restreints par manque de jeux d'essais, et que la différence théorique soit très difficile à évaluer : comme on l'a vu plus haut, selon les caractéristiques de la grammaire minimaliste employée, la conversion (et, selon toute vraisemblance, l'analyse suivant quelque implémentation que ce soit) peut atteindre une complexité exponentielle.

Etant donnée l'aisance relative avec laquelle une grande partie de l'humanité est capable de s'exprimer, nous pouvons raisonnablement supputer que, si jamais les grammaires minimalistes sont bien le formalisme utilisé inconsciemment pour la représentation des langues humaines, celles-ci restent simples...

Par exemple, dans [Sta97a], Edward Stabler formalise une hypothèse restreignant à vingt-quatre possibilités la partie syntaxique des entrées lexicales. En composant avec ces possibilités, il devrait être possible de représenter essentiellement toutes les langues. Ces entrées utilisent la variante des grammaires minimalistes avec traits forts et faibles, ce qui ne change pas foncièrement la complexité de la conversion, et chaque entrée comporte, au plus, deux sélecteurs, un trait de mouvement et quatre traits au total. Selon nos estimations, la conversion d'une telle grammaire, d'un lexique comportant n entrées, serait au plus, de l'ordre de $1.2 \times 10^{11} + O(n)$ opérations (selon l'analyse en temps), ou $8.5 \times 10^6 + O(n)$ règles (selon l'analyse en taille), et sans doute avec des constantes sensiblement inférieures, ce qui correspond plus à l'ordre de grandeur attendu pour une analyse de

langue humaine...

Au vu des équivalences présentées plus haut, une expérimentation intéressante pourrait être de comparer les performances d'une conversion en MCFG, suivie d'une application de l'analyseur d'Albro d'une part, et suivie d'une conversion (triviale) en une sPRCG G_{RCG} et de l'application de l'analyseur de type Earley pour les RCG. Les résultats devraient être comparables, cette dernière analyse étant également en $O(n^p)$ pour une phrase de longueur n , avec un p dépendant de caractéristiques de la grammaire¹.

¹D'après [Bou98], cette complexité est plus précisément, au pire, de $O(k(n^{2h(l+1)}))$, pour k le nombre de règles de G_{RCG} , h sa dimension maximale (G_{RCG} est une h-sPRCG quand G_{MCFG} est une h-MCFG), et l la longueur maximale d'une règle, en terme de prédicats utilisés en partie droite.

Chapitre 5

Conclusion

My name, and yours, and the true name of the sun, or a spring of water, or an unborn child, all are syllables of the great word that is very slowly spoken by the shining of the stars.

U. K. Le Guin

Le premier objet de ce travail était de cataloguer et de présenter plusieurs formalismes du traitement automatique des langues pour la représentation syntaxique. En effet, pour un chercheur, qu'il soit linguiste, psychologue, neurologue, informaticien ou mathématicien, le domaine de la linguistique computationnelle peut sembler très ardu au premier abord, eu égard au grand nombre de concepts spécifiques à cette discipline.

Dans cette optique, la première partie de ce mémoire peut permettre de retrouver rapidement les caractéristiques principales, notamment en termes de pouvoir d'expression, d'un formalisme donné de traitement automatique des langues qu'on peut avoir à étudier ou approfondir.

Ce tableau des formalismes est loin d'être exhaustif, mais nous nous sommes attachés à présenter, pour chaque palier de classes de langages, certains de ses représentants les plus étudiés. Nous avons aussi cherché à effectuer un état de l'art des équivalences entre formalismes, en présentant certains des travaux de référence dans ce domaine.

Le sujet proposait ensuite d'étudier particulièrement les grammaires minimalistes et les formalismes qui leur sont équivalents. Il s'avère que les équivalences, procédures de conversion, ainsi que les algorithmes d'ana-

lyse qui en résultent ont récemment fait l'objet d'études approfondies, mais sans doute encore peu connues par la communauté de la linguistique computationnelle. Une partie de ce mémoire est consacrée à la présentation et à l'explicitation, d'une manière se voulant pédagogique, de ces travaux en particulier, de façon à ce qu'ils soient rendus plus accessibles.

Nous avons aussi tenté d'évaluer la pertinence d'objections pouvant être formulées quant à la performance des algorithmes d'analyse que ces études ont permis de dégager, et, ce faisant, de comprendre les facteurs pouvant accroître la complexité d'analyse ou de génération de textes suivant les instances particulières des formalismes utilisés – car, même s'il apparaît que les langues utilisées dans la communication humaine sont bien incluses dans les langages pouvant être exprimés par certains des formalismes étudiés (avec sans doute quelques aménagements), elles semblent n'en former qu'une très petite partie. . .

Au cours du stage de recherche ayant donné lieu à la rédaction de ce mémoire, nous avons été amenés à nous interroger sur l'évolution du traitement automatique des langues.

Il apparaît que la problématique liée à l'analyse syntaxique de la phrase et à la modélisation de la compétence de génération de phrases à partir de grammaires est en voie de résolution : que ce soit à partir d'un formalisme dérivé des grammaires minimalistes, des grammaires à concaténation d'intervalles, ou d'un nouveau formalisme, probablement composé de l'un ou de l'autre et modulé par des résultats d'études de psycho- ou de neuro-linguistique, il semble qu'un (voire de multiples) formalisme unifié, capable de représenter la syntaxe de l'ensemble des langues appréhendables par l'esprit humain et qu'il soit possible de mettre en œuvre par des algorithmes simples, soit en passe d'être achevé.

Il ne restera alors plus qu'à effectuer l'immense travail consistant à modéliser, au moyen de grammaires instanciant ce formalisme, chacun des phénomènes linguistiques existants, l'ensemble des mots du vocabulaire et des règles de grammaire et d'orthographe les liant, ainsi que les exceptions à ces règles, et, dans le cas particulier de la langue française, les exceptions aux exceptions. . .

De notre point de vue, il est désormais important de chercher à modéliser les aspects de la langue allant au-delà de la syntaxe de la phrase.

En effet, la recherche en traitement automatique des langues est bien moins avancée dans ces domaines, qui font pourtant tout autant partie de la compétence linguistique du locuteur : qu'il s'agisse de la *sémantique*, qui

permet de dégager, parallèlement à la structure syntaxique de la phrase (qui organise les mots), sa structure logique (qui organise les concepts) ou de la *pragmatique*, qui situe le contexte de chaque élément par rapport aux informations supposées connues, de nombreux mécanismes restent à mettre en place pour permettre de modéliser le *sens* des mots, qu'il soit *dénotatif* (la définition du dictionnaire – qui peut déjà être ambiguë) ou *connotatif* (représentant, entre autres les sens associés par similarité logique ou phonétique, les figures de style argotiques, prosodiques ou poétiques), et à les assembler pour former le sens de la phrase, de l'énoncé, ou du texte.

Ces travaux devront s'appuyer sur une analyse syntaxique robuste, pouvant être fournies par les grammaires minimalistes : en effet, l'organisation de la syntaxe reste la base des relations entre les concepts représentés par les mots ; il serait dommage de connaître le sens de tous les mots d'un texte, mais de ne pas pouvoir les combiner.

De nombreuses recherches sont en cours, d'autres restent à effectuer, dans ces domaines, avant d'atteindre la modélisation informatique complète de la *compréhension* de la langue...

Annexe A

Glossaire des acronymes de formalismes

CFG : Pour *Context-Free Grammars*, ou grammaires hors contexte. Formalisme classique, parmi les grammaires génératives, engendrant les langages de type 2. Voir la hiérarchie de Chomsky, section 2.1.1, p. 9.

GCFG : Pour *Generalized Context-Free Grammars*, ou grammaires hors contexte généralisées. Extension des CFG, portant sur des tuples de chaînes, chaque non-terminal étant produit par l'application d'une fonction quelconque d'autres non-terminaux. Définition : section 3.2.1, p. 19. Présentées dans [SMFK91].

HG : Pour *Head Grammars*, ou grammaires de tête. Extension des CFG, portant sur des couples de chaînes. Elles ne sont pas, ici, définies en détail. Leur pouvoir d'expression est équivalent aux TAG. – voir fig. 2.2, p. 12.

hMG : Ou grammaires minimalistes augmentées. Variantes des MG comportant plusieurs fonctionnalités supplémentaires, comme le mouvement de tête. Des règles de dérivation supplémentaires sont ajoutées, les traits peuvent différer, mais, pour les variantes regroupées sous cette appellation, la classe de langages n'est pas modifiée. Éléments de définitions : section 3.4.4, p. 32. Formalismes faiblement équivalents : MG, MCFG, LCFRS, MCTAG, sPRCG.

LCFRS : Pour *Linear Context-Free Rewriting Systems*, ou systèmes de

réécriture hors contexte linéaires. Variante des GCFG restreignant la portée des fonctions utilisées par les règles, n'autorisant ni la copie, ni l'effacement de variables. Définition et exemples : section 3.2.4, p. 23. Formalismes équivalents : MG, hMG, MCTAG, MCFG, sPRCG. L'équivalence (forte) LCFRS-sPRCG est détaillée en section 4.1.2, p. 34, ainsi que par [Bou98], et l'équivalence LCFRS-MCFG est détaillée en section 4.1.3, p. 35.

MCFG : Pour *Multiple Context-Free Grammars*, ou grammaires hors contextes multiples. Variante des GCFG restreignant la portée des fonctions utilisées par les règles, autorisant l'effacement, mais non la copie, de variables. Définition et exemples : section 3.2.3, p. 22. Présentées dans [SMFK91]. Formalismes faiblement équivalents : MG, hMG, LCFRS, sPRCG, MCTAG. L'équivalence MG-MCFG est détaillée section 4.2, p. 35.

MCTAG : Pour *Multiple Component TAG*, ou grammaires à adjonction d'arbres, à composantes multiples. Variantes des TAG autorisant de multiples possibilités d'adjonction pour un arbre auxiliaire donné. Définition : section 3.1.1, p. 18. Formalismes faiblement équivalents : MG, hMG, LCFRS, MCFG, sPRCG.

MG : Pour *Minimalist Grammars*, ou grammaires minimalistes. Grammaires génératives fonctionnant par mouvement et combinaison d'expressions, nées du programme minimaliste de Noam Chomsky. Définitions et exemples : section 3.4, p. 26. Formalismes faiblement équivalents : hMG, MCTAG, sPRCG, LCFRS, MCFG (l'équivalence MG-MCFG est détaillée section 4.2, p. 35).

PMCFG : Pour *Parallel Multiple Context-Free Grammars*, ou grammaires hors contexte multiples parallèles. Variante des GCFG restreignant leur portée, permettant la copie ou l'effacement des variables. Définition et exemples section 3.2.2, p. 20. Présentées dans [SMFK91]. Formalismes équivalents : aucun, mais, une fois étendues pour pouvoir être closes par intersection, occupent la même classe que les RCG – voir section 4.1.1, p. 34.

RCG : Pour *Range Concatenation Grammars*, ou grammaires à concaténation d'intervalles. Adaptées des systèmes grammaticaux à contraintes

utilisés, par exemple, dans PROLOG, et permettent d'exprimer exactement tous les langages analysables en temps polynomial (avec un exposant arbitraire). Définition et exemples : section 3.3, p. 25. Présentées dans [Bou98]. Formalismes faiblement équivalents : PMCFG avec intersection, voir section 4.1.1, p. 34.

sPRCG : Pour *simple positive RCG*, ou grammaires à concaténation d'intervalles, simples et positives. Variante des RCG n'utilisant que des prédicats, positifs, sans effacement, copie, ou utilisation de terminaux en partie droite des règles. Présentées dans [Bou98]. Définition et exemples : section 3.3.1, p. 25. Formalismes faiblement équivalents : MG, hMG, MCFG, MCTAG, LCFRS (l'équivalence LCFRS-sPRCG est forte, et détaillée en section 4.1.2, p. 34).

TAG : Pour *Tree Adjoining Grammars*, ou grammaires à adjonction d'arbres. Permettent de représenter de façon intuitive la structure arborescente de la phrase. Définition et exemples : section 3.1, p. 15. Définies par [JLT75]. Formalismes faiblement équivalents : HG.

Annexe B

Références bibliographiques et Internet

Les algorithmes de conversion des grammaires minimalistes (MG comme hMG) de [Gui04] et d'analyse des MCFG ont été récemment implémentés par Peter Ljunglöf sous forme de librairie Prolog, disponible à <http://www.cs.chalmers.se/~peb/software.html>.

Plusieurs utilitaires, dont les analyseurs brièvement évoqués ici, sont indiqués sur la page d'Edward Stabler, <http://www.linguistics.ucla.edu/people/stabler/epssw.htm>.

Sur cette page sont également indiqués de nombreux autres travaux de référence, ainsi que l'état de la recherche actuelle.

Ainsi, les travaux de Jens Michaelis, portant sur les propriétés formelles des grammaires minimalistes et de leurs variantes, sont disponibles à <http://tcl.sfs.uni-tuebingen.de/~michael/papers.html>.

Les publications de Maxime Amblard, dont les recherches portent sur une interface syntaxe-sémantique pour les grammaires minimalistes, sont disponibles à <http://www.labri.fr/perso/amblard/>.

L'analyseur de structure logique de la phrase, conçu et proposé par Richard Moot, qui peut être une base syntaxique pour des études sémantiques ou pragmatiques, *Grail*, est disponible à <http://www.labri.fr/perso/moot/grail.html>.

Bibliographie

- [Alb00] Daniel M. Albro. An earley-style recognition algorithm for mcfgs. 2000.
- [Amb03] Maxime Amblard. Représentations sémantiques pour les grammaires minimalistes. Technical report, Université Bordeaux 1 / INRIA, June 2003.
- [Bou98] Pierre Boullier. Proposal for a Natural Language Processing Syntactic Backbone. Technical report, INRIA, January 1998.
- [Bou99] Pierre Boullier. Chinese numbers, mix, scrambling, and range concatenation grammars. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 53–60, Bergen, Norway, June 1999.
- [Ear70] J. Earley. An efficient context-free parsing algorithm. In *Communications of the Association for Computing Machinery*, 1970.
- [Gui04] Matthieu Guillaumin. Conversions between Mildly Context Sensitive Grammars. 2004.
- [Har01] Hendrik Harkema. Parsing Minimalist Languages. 2001.
- [JLT75] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 1975.
- [Jos] Aravind K. Joshi. Mildly Context-Sensitive Grammars. <http://www.kornai.com/MatLing/mcsfin.pdf>.
- [Jos85] Aravind K. Joshi. Tree Adjoining Grammars : How much context sensitivity is required to provide a reasonable structural description. *Natural Language Parsing*, pages 206–250, 1985.
- [Kas65] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA, 1965.

- [KMMM03] Hans-Peter Kolb, Jens Michaelis, Uwe Mönnich, and Frank Morawietz. An operational and denotational approach to non-context-freeness. *Theor. Comput. Sci.*, 293(2) :261–289, 2003.
- [KS94] Edward Keenan and Edward Stabler. There is more than one language. In *Langues et Grammaire-I*, 1994.
- [Lju05] Peter Ljunglöf. A Polynomial Time Extension of Parallel Multiple Context-Free Grammar. In *LACL 2005*, 2005.
- [Mic01a] Jens Michaelis. Derivational minimalism is mildly context-sensitive. In *LACL '98 : Selected papers from the Third International Conference, on Logical Aspects of Computational Linguistics*, pages 179–198, London, UK, 2001. Springer-Verlag.
- [Mic01b] Jens Michaelis. Observations on Strict Derivational Minimalism. In *Proceedings of the joint meeting of the 6th Conference on Formal Grammar and the 7th Conference on Mathematics of Language (FGMOL '01)*, 2001.
- [Mic01c] Jens Michaelis. *On Formal Properties of Minimalist Grammars*. PhD thesis, Postdam, Germany, 2001.
- [Mic01d] Jens Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In *LACL '01 : Proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics*, pages 228–244, London, UK, 2001. Springer-Verlag.
- [Mic02] Jens Michaelis. Implications of a revised perspective on minimalist grammars, 2002.
- [MK05] Jens Michaelis and Greg Kobele. Two Type 0-Variants of Minimalist Grammars. In *The 10th conference on Formal Grammar and The 9th Meeting on Mathematics of Language*, August 2005.
- [SCK⁺03] Edward P. Stabler, Travis C. Collier, Gregory M. Kobele, Yoonsok Lee, Ying Lin, Jason Riggle, Yuan Yao, and Charles E. Taylor. The learning and emergence of mildly context sensitive languages. In *European Conference of Artificial Life*. ECAL, 2003.
- [SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theor. Comput. Sci.*, 88(2) :191–229, 1991.

- [Sta97a] Edward P. Stabler. Acquiring languages with movement. Technical report, University of California Los Angeles, 1997.
- [Sta97b] Edward P. Stabler. Derivational minimalism. *Lecture Notes in Computer Science*, 1328 :68–??, 1997.
- [Sta00] Edward P. Stabler. Minimalist grammars and recognition. Technical report, University of California Los Angeles, 2000.
- [Sta01] Edward P. Stabler. Recognizing head movement. In *LACL '01 : Proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics*, pages 245–260, London, UK, 2001. Springer-Verlag.
- [Sta05] Edward P. Stabler. Languages universals : a computational perspective. In *LSA 2005*, 2005.
- [Stu85] Stuart M. Shieber. Evidence againts the context-freeness of natural language. In *Linguistics and Philosophy 8*, pages 333–343. D. Reidel Publishing Company, 1985.
- [Van93] GJM VanNoord. *Reversibility in Natural Language Processing*. PhD thesis, University of Utrecht, 1993.
- [vV96] N. van Vugt. *Generalized Context-Free Grammars*. PhD thesis, Leiden, The Netherlands, 1996.

Annexe C

Remerciements

Ce travail n'aurait pas pu avoir lieu sans le soutien indéfectible de ma famille et de mes amis, notamment de Samuel Méril et Frank Morfea, expatriés, de Michel Fournier, victime malheureuse de la réduction des postes budgétaires à l'Université, de Nada Ayad, exilée dans une formation d'ingénieurs, ainsi que de mes collègues, étudiant dans d'autres formations (telles que la physique quantique ou la biochimie), qui m'ont fait prendre conscience de la relative simplicité de ma tâche. Je tiens également à remercier le personnel et le café de la Soucoupe, cafétéria universitaire qui, en l'absence de local réservé aux stagiaires en mémoire de recherche, est rapidement devenue mon lieu de travail privilégié.

Les rencontres et longues discussions hebdomadaires avec mes encadrants : Maxime Amblard, Irène Durand et Christian Retoré, ont été les moments les plus précieux de ce semestre de près de quatre mois. Sans leurs interrogations, conseils, suggestions, ce travail aurait été de beaucoup écourté. C'est par cette interaction constante, ce travail en collaboration, que ce mémoire a pu véritablement voir le jour, même s'il n'a pas répondu à toutes les attentes du sujet posé au départ – sujet qui, nous le savons désormais, était très ambitieux pour un mémoire d'un semestre.

D'autres personnes ont contribué, à la réalisation de ce mémoire : MM. Kasami, Seki, Fuji et Matsumura, de par leur article très complet et instructif [SMFK91] ; MM. Stabler et Michaelis, pour leurs travaux et leur disponibilité, en particulier.

Je me dois aussi de citer D. E. Knuth, Jeffrey Mark Siskind, toute la Free Software Foundation et toutes les équipes d'Apple Computers pour les logiciels utilisés ; la classe politique française, qui s'est provisoirement rendue à la raison, me libérant provisoirement de mon devoir de revendication ; Clamp, Yôko Kanno et de nombreux artistes et musiciens (en grande partie originaires du Japon), les œuvres desquels m'ont constamment accompagné ; et sans doute plusieurs centaines de personnes injustement oubliées...

Bruno Mery, 12 Juin 2006.