

*A Case Study of the Convergence of
Mildly Context-Sensitive Formalisms for
Natural Language Syntax:
from Minimalist Grammars
to Multiple Context-Free Grammars*

Bruno Mery — Maxime Amblard — Irène Durand — Christian Retoré

N° 6042

September 2006

Thème SYM



*Rapport
de recherche*



**A Case Study of the Convergence of
Mildly Context-Sensitive Formalisms for
Natural Language Syntax:
from Minimalist Grammars
to Multiple Context-Free Grammars**

Bruno Mery , Maxime Amblard , Irène Durand , Christian Retoré

Thème SYM — Systèmes symboliques
Projet Signes

Rapport de recherche n° 6042 — September 2006 — 58 pages

Abstract: The present work is set in the field of natural language syntactic parsing. We present the concept of "mildly context-sensitive" grammar formalisms, which are full-fledged and efficient for syntactic parsing. We summarize a number of these formalisms' definitions, together with the relations between one another, and, most importantly, a survey of known equivalences. The conversion of Edward Stabler's Minimalist Grammars into Multiple Context-Free Grammars (MCFG) is presented in particular detail, along with a study of the complexity of this procedure and of its implications for parsing. This report is an adaptation of the French Master thesis that bears the same name, from Bordeaux 1 University, June 2006.

Key-words: formal grammar, formal language theory, mildly context-sensitive grammars, minimalist grammars, natural language syntax, parsing

**Étude de la Convergence des
Formalismes Légèrement Contextuels pour
la Syntaxe des Langues :
des Grammaires Minimalistes
aux Grammaires Algébriques Multiples**

Résumé : Ce travail porte sur l'analyse syntaxique des langues. Il aborde la notion de formalismes grammaticaux "légèrement contextuels", en rappelant leurs avantages pour une analyse syntaxique complète et efficace. Plusieurs de ces formalismes sont présentés, ainsi que les relations qui les unissent ; on fournira notamment un état des équivalences connues. La conversion des Grammaires Minimalistes de Edward Stabler en Grammaires Algébriques Multiples (MCFG) est particulièrement détaillée ; on étudie aussi la complexité de cette procédure et les perspectives induites pour l'analyse syntaxique de la phrase. Ce rapport est tiré du mémoire de Master du même nom, soutenu à l'Université de Bordeaux 1 en Juin 2006.

Mots-clés : grammaires formelles, théorie des langages, grammaires légèrement contextuelles, grammaires minimalistes, analyse syntaxique du langage naturel

Contents

| | | |
|----------|--|-----------|
| 1 | Foreword | 1 |
| 2 | Situation of the various formalisms : an overview | 3 |
| 2.1 | The relative place of the formalisms | 3 |
| 2.1.1 | Chomsky's Hierarchy, regular, context-free and context-sensitive languages | 3 |
| 2.1.2 | Mildly context-sensitive languages | 5 |
| 2.2 | Hierarchy of the formalisms | 7 |
| 2.2.1 | Neighbor classes of TAGs | 7 |
| 2.2.2 | Neighbor classes of Minimalist Grammars | 8 |
| 3 | Formal definitions and examples | 11 |
| 3.1 | Tree adjoining grammars | 11 |
| 3.1.1 | Multiple component tree adjoining grammars | 13 |
| 3.2 | Generalized context-free grammars and their restrictions | 15 |
| 3.2.1 | Generalized context-free grammars | 15 |
| 3.2.2 | Parallel k -multiple context-free grammars | 16 |
| 3.2.3 | k -multiple context-free grammars | 18 |
| 3.2.4 | Linear context-free rewriting systems | 19 |
| 3.3 | Range concatenation grammars | 19 |
| 3.3.1 | Simple positive range concatenation grammars | 21 |
| 3.4 | Minimalist grammars | 21 |
| 3.4.1 | Features and the lexicon | 22 |
| 3.4.2 | Simple minimalist grammars | 23 |
| 3.4.3 | General minimalist grammars | 25 |
| 4 | Equivalences and conversion of the formalisms | 27 |
| 4.1 | Variations of PMCFGs, MCFGs and RCGs | 27 |
| 4.1.1 | PMCFGs and RCGs | 27 |
| 4.1.2 | LCFRSs and sPRCGs | 27 |
| 4.1.3 | MCFGs in normal form | 28 |
| 4.2 | MG-MCFG equivalence and minimalist grammars parsing | 29 |

| | | |
|----------|--|-----------|
| 4.2.1 | The procedure of converting minimalist grammars into MCFGs | 29 |
| 4.2.2 | Conversion example | 33 |
| 4.2.3 | Conversion complexity | 37 |
| 4.2.4 | Parsing minimalist grammars | 39 |
| 5 | To sum up... | 43 |
| A | Tree structures vs. String manipulations | 45 |
| B | A Glossary of the formalism acronyms used | 49 |
| C | Internet and bibliographical references | 51 |
| D | Acknowledgments | 57 |

List of Figures

| | | |
|-----|---|---|
| 2.1 | Chomsky's Hierarchy: place of mildly context-sensitive languages | 6 |
| 2.2 | Hierarchy of some mildly context-sensitive formalisms, of generative power neighboring context-free languages | 7 |
| 2.3 | Hierarchy of some mildly context-sensitive languages, of generative power neighboring minimalist grammars | 8 |

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

Chapter 1

Foreword

Although studies on the nature of human languages have been undertaken during History throughout, the interest of linguistics in the structures that may model these languages appeared only rather recently (circa 1950). This work has become greatly needed when computer scientists, attempting to implement automated translation, were stopped by the unforeseen, unmeasurable complexity of the languages. It quickly appeared that, in order to account for every linguistic phenomenon, new formalisms had to be constructed.

The understanding, by a computer, of the meaning of an arbitrary utterance is a problem that will most probably prove very long and difficult to solve completely : it is thought to be *AI-complete*, that is to say that its solution entails the creation of a system able to pass the Turing test, i.e. as intelligent as a human (some advances in this field are indeed based upon the analysis of human thought processes). Nevertheless, the automatic analysis of specific parts of the syntax, grammar or semantics of languages has already numerous applications. Indeed, the extraction of information and data mining, translation of texts in particular cases (such as technical manuals) and assisted translation, typographical and (to some extent) spelling correction, question answering, generation of random texts (such as Raymond Queneau experimented with), are many fields for which research in computational linguistics and natural language processing has contributed.

As part of this research, numerous mathematical formalisms, originating from computer science and linguistics, were theorized. Their ultimate goal is to model the whole set of the linguistic processes used in human communication, remaining simple enough to implement and efficient enough for everyday use. We will detail the existing formalisms that are for now the most adequate candidates for this goal, restricting our approach to the *generative power*. This notion models the ability to generate, in a structured way, the relevant languages.

Thus, the other aspects of natural language processing are left out of this report. These separate aspects include, e.g., the generation, for a given sentence, of a logical formula modeling its semantical sense, or the use of semantical or pragmatical knowledge. . . Those are so many dedicated study subjects.

From the point of view of linguistics, the structure of a sentence as it is given by these formalisms corresponds to the *syntax* of the sentence, that is to say, its organization regarding the language's grammatical rules. The goal is to model the *skill* of the speaker, i.e., her ability to parse the syntax of an arbitrary long sentence, given the rules of the grammar and the vocabulary of the language. A study based on her *performance*, on the other hand, assumes that utterances are relatively small. For instance, a sentence has a finite and quite small number of words (a few tens at most). The syntactical constructs are also limited, e.g., a sentence seldom contains more than three imbricated relative subordinates. Such an approach would easily allow to describe languages as they are usually spoken, but not the processes that enable us to handle them. . .

In that way, *context-free grammars*, which are akin to the rules of grammar taught in third grade¹, are a mathematically simple and very efficient formalism, and can actually be used for restricted languages (such as programming languages), but are nevertheless insufficient for human languages.

On the other hand, other formalisms can express any recursively enumerable language but are not focused enough. They cannot be restricted to some target linguistic phenomena and generate languages much more complex than human languages. Nor are they efficient enough, the problem of the ownership of a given sentence to a language using these formalisms being at best NP-complete.

One of the theoreticians' goals was, as time went by, to find a formalism approaching a minimal bound, i.e., taking into account exactly all linguistic phenomena.

Two conjectures on these formalisms have been induced from various studies in formal linguistics. First, such formalisms must allow for parsing sentences in a reasonable time, lest the speakers not understand a sentence of more than a few words quickly enough for a talk (which is usually thought of as polynomial time parsing or better). Second, there must exist a learning algorithm for a given language, generated by the mean of any of these formalisms, from positive utterances – i.e., one must be able to derive a description of the language from a number of correct sentences, as young children do.

A characterization, introduced by Joshi, calls the formalisms within those constraints, along with the generated languages, *mildly context-sensitive*. We will detail that characterization, together with the place of the languages generated by these, and then the formalisms themselves, with a study of the known equivalences between one another. We will focus on *minimalist grammars*, which result of the synthesis of linguistic hypotheses over language universality, and on *range concatenation grammars*, possessing interesting formal properties, along with a few formalisms in between.

¹On the form : a *noun phrase* can be a *proper noun*, a *determiner* followed by a *noun*, or a *determiner* followed by an *adjective* and a *noun*. . .

Chapter 2

Situation of the various formalisms : an overview

2.1 The relative place of the formalisms

In this section, we will introduce the formalisms that we shall study, their place and their relationships, as well as relationships to other classical formalisms. In the present report, they will be classified according to their *generative power*, i.e., the class of string languages which they generate.

We will indeed focus on the *competence* of the speaker: the ability, for an human knowing the vocabulary and the grammatical rules of a language, to produce correct sentences (including ones that she never heard before), and to be able to tell for a given sentences (that she may not have known) is grammatically correct or not. As we said, we are not concerned with her *performance*.

2.1.1 Chomsky's Hierarchy, regular, context-free and context-sensitive languages

Noam Chomsky, very influential reforming linguist and political trouble-maker since 1955, is the creator of the theory of *generative grammar*. The different properties of these grammars, which generate very varied languages when different constraints are enforced, have allowed him to define a language hierarchy which has become a reference for all other formalisms.

A generative grammar is defined by a set N of non-terminal symbols (denoted A, B, \dots), a set T of terminal symbols, that represents the words or idioms of the language¹ (denoted $a, b, \dots - T$ is also the alphabet from which the generated language will be written), and a set of rules of the form

¹For different uses, they may also represent letters, or, e.g., in morphological studies, elementary morphemes or phonemes, from which words may be derived by flexion, and so on.

$\alpha \rightarrow \beta$, where $\alpha \in (N \cup T)^* N (N \cup T)^*$, $\beta \in (N \cup T)^*$. Given an *axiom* (or *start symbol*) $S \in N$, the generated language is the set of strings of terminals γ such as $S \rightarrow^* \gamma$.

For instance, the rules of English described by "a noun phrase (*NP*) can be a proper noun (*PN*), a determiner (*D*) followed by a noun (*N*), or a determiner followed by an adjective (*A*) and a noun" would be formalized by the following generative grammar rules :

$$\begin{aligned} NP &\rightarrow PN \\ NP &\rightarrow D N \\ NP &\rightarrow D A N \end{aligned}$$

The languages of Chomsky's Hierarchy are distinguished into four classes, named "type-0", "type-1", "type-2" and "type-3", with:

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

There also exist non recursively enumerable languages, that do not possess any of the properties characterizing those classes, and are seldom studied².

Those four classes are defined as follows:

- type-0:** The set of *recursively enumerable* languages. Numerous formalisms allow for their construction, including Turing machines and unrestricted generative grammars. This set is however too large for automated processing, and the analysis of those languages is a problem NP-complete at least.
- type-1:** *Context-sensitive* languages. By contrast with more restricted language classes, context-sensitive languages require knowledge of some data beyond the scope of the phrase being analyzed, its *context*³. For any language in this class, the problem of knowing whether a string is generated by this language is *decidable*.
- type-2:** *Context-free* languages. Those languages are generated and characterized by the set of *Context-Free Grammars* (CFG), which have been studied extensively for their properties (noteworthy, a word or sentence of n symbols of a given context-free language, whose grammar is known, can be parsed in $O(n^3)$ time).
- type-3:** *Regular* languages, constructed and recognized by finite state automata or regular expressions, among other formalisms. These are very useful for very specific applications, for they are very efficient, but are too restricted for most language processing problems (every known human language being non-context-free and, of course, not regular).

²In fact, type-0 languages being *Turing-complete*, and therefore containing every language that could be described by an arbitrary algorithmical formalism, there cannot be any useful description of non recursively enumerable languages. The only proof that might be given of their being is that their non-existence would lead to a contradiction. One can then easily understand that this particular language class does only moderately concern linguists...

³A *context-sensitive grammar* would be analogous to construction rules of this form: a *verb phrase*, when followed by an *complement*, can be constituted by a *subject phrase* and a *transitive verb*. The presence of the complement is an information which is not contained in the verbal phrase, but rather in its *context*.

2.1.2 Mildly context-sensitive languages

Individually, most linguistic phenomena can be modeled by context-free languages. However, many works, among them [Shi85], have shown that human language as a whole is beyond the generative power of this language class. In particular, phenomena such as cross-dependencies are beyond their reach⁴.

Various arguments, including the need of a quick analysis and the possibility to be *learnt*, give rise to think that human languages are included in context-sensitive languages. In [Jos85], Joshi proposes a characterization for what he calls *Mildly Context-Sensitive Languages* (MCSLs), aiming to be the smallest possible language class that include all linguistic phenomena. This class is the set of all languages with the following properties, succinctly detailed in [Jos]:

1. Context-free languages are strictly included in MCSL.
2. Any language in MCSL can be analyzed in polynomial time.
3. The languages in MCSL may represent some linguistic phenomena. That particular criterion is intentionally loose. The author suggests, for instance, to consider cross-references and nested dependencies in some germanic languages, but only to a certain extent.
4. Languages in MCSL have the *constant growth* property.

This definition is ambiguous by design. One usually thinks of *Tree Adjoining Grammars* (TAGs) and the richer *Multiple Context-Free Grammars* (MCFGs), for example, as generating mildly context-sensitive languages.

⁴Thus, in English, a sentence on the model of $A_1, A_2 \dots$ and A_n , who respectively dwell in $B_1, B_2 \dots$ and B_n is akin, in form, to $\{w^2\}$, which is not a context-free language. Numerous studies have been undertaken in order to find more frequent non-context-free structures in Dutch, or, as in [Shi85], in German Swiss.

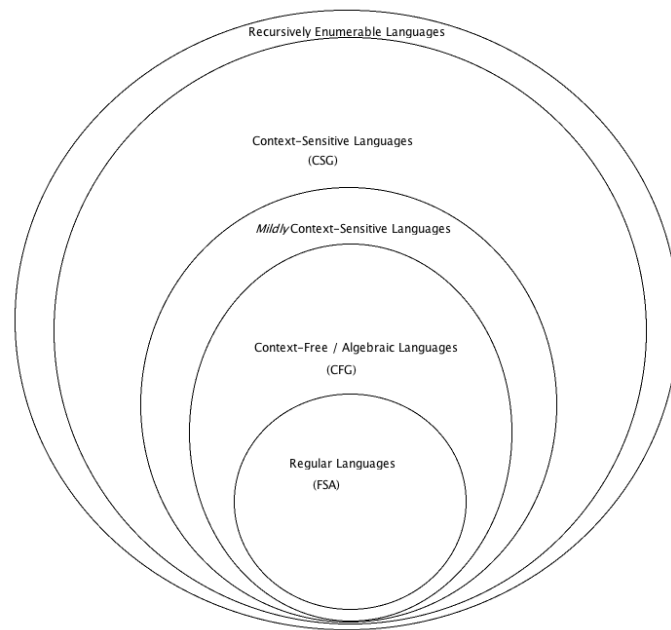


Figure 2.1: Chomsky's Hierarchy: place of mildly context-sensitive languages

2.2 Hierarchy of the formalisms

We will discuss the relative place of some of the most commonly used formalisms. Among these, some of particular interest to this study will be defined and detailed thereafter.

2.2.1 Neighbor classes of TAGs

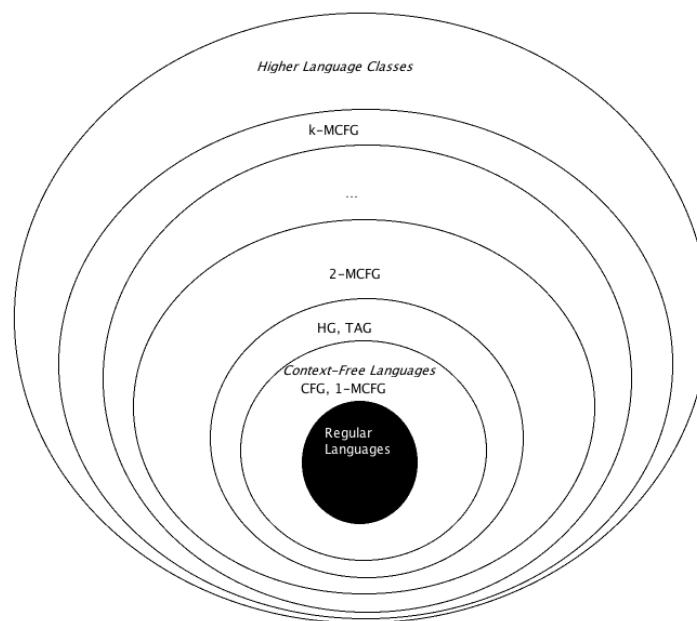


Figure 2.2: Hierarchy of some mildly context-sensitive formalisms, of generative power neighboring context-free languages

k -multiple context-free grammars (k -MCFGs) are formalisms based upon context-free grammars. 1 -MCFGs are by definition equivalent to CFGs, and $k+1$ -MCFGs strictly contains k -MCFGs. The definitions of these formalisms, and the proofs of the equivalences, are given in [SMFK91].

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

Head Grammars (HG) and Tree Adjoining Grammars (TAGs) are commonly studied formalisms, whose string language classes are equivalent⁵, and strictly included in that of 2-MCFGs. These equivalences and inclusions are also proved in [SMFK91].

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

2.2.2 Neighbor classes of Minimalist Grammars

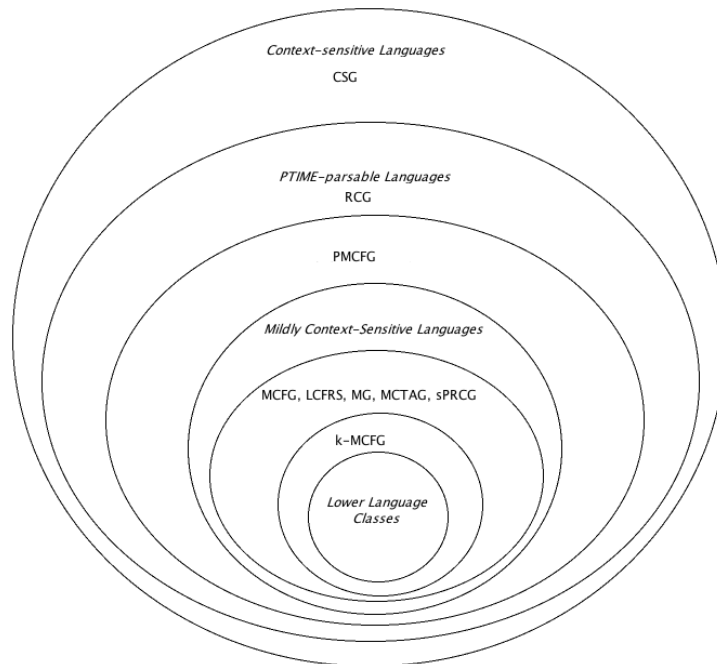


Figure 2.3: Hierarchy of some mildly context-sensitive languages, of generative power neighboring minimalist grammars

⁵While we review some of the most known formalisms, there are many others that could be included in a complete study. For example, [Bou98] proves that Linear Indexed Grammars (LIGs) and 2-sPRCGs (sPRCGs with at most two arguments – see the next session for an overview of generalized sPRCGs and section 3.3.1, p. 21 for a formal definition) are both equivalent to HGs and TAGs, and are therefore also part of that particular class.

An important class of formalisms is the one that shares string languages with MCFGs (i.e., k -MCFGs for any given k). It includes:

- MCFGs, Linear Context-Free Rewriting Systems (LCFRSs, which are defined in the same fashion),
- Multiple Component Tree Adjoining Grammars (MCTAGs, a common extension of the aforementioned TAGs),
- simple Positive Range Concatenation Grammars (sPRCGs, a particular case of Range Concatenation Grammars, both introduced by Pierre Boullier in [Bou98]), and
- Minimalist Grammars (MGs, introduced by Edward Stabler in order to implement Chomsky's Minimalist Program of the \bar{X} -theory, with which we are concerned most in this study).

The equivalence between the similar MCFGs and LCFRSs is well known, and detailed in [SMFK91].

The equivalence between many formalisms, including MCTAGs and LCFRSs, is detailed in [Wei88]. Pierre Boullier, in his survey article [Bou98], proves the equivalence between LCFRSs and sPRCGs.

And finally, Jens Michaelis has formally studied the generative power of MGs and, in two articles, [Mic01d] and [Mic01a], demonstrates the equivalence between LCFRSs and minimalist grammars. [Har01b] contributed similar results at the same time.

All languages included in the class common to all of these formalisms are thought of as mildly context-sensitive, being parsable in polynomial time and of constant growth.

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

Parallel Multiple Context-Free Grammars (PMCFGs, an extension of MCFGs) have a generative power larger than MCFGs, as demonstrated by [SMFK91].

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

Range Concatenation Grammars (RCGs), formerly introduced in [Bou98], are a generalization of sPRCGs. In the same article, their relationship to another less-known formalism, Literal Movement Grammars (LMGs), and its variations is studied. LMGs and their variants, simple-LMGs and index-LMGs, are introduced in [Gro97], as string manipulation formalisms, and we will not investigate them further in this work. The interesting result is that index-LMGs and simple-LMGs are both equivalent, and they generate exactly all polynomial-time (with an arbitrary exponent) parsable languages. [Bou98] uses this result, and points out the equivalence between simple-LMGs and RCGs, to prove that RCGs cover exactly *PTIME* (as this language class is formally called).

Human languages being considered understandable (i.e., easily parsable), they are included in *PTIME*.

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

Moreover, PMCFG and RCG generate more than mildly context-sensitive languages, e.g., their generative power includes the language $\{a^{(2^n)}\}$.

Finally, Generalized Context-Free Grammars, from which MCFGs are derived, generate exactly the set of recursively enumerable languages, including context-sensitive languages. That characterization is given in [SMFK91].

$$MCSL \subsetneq CSL \subsetneq REL = L(GCFG)$$

Chapter 3

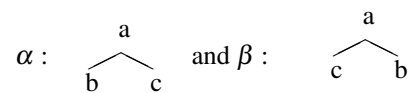
Formal definitions and examples

3.1 Tree adjoining grammars

Tree adjoining grammars (TAGs) were introduced by Joshi in [JLT75] and are one of the earliest examples of mildly context-sensitive formalisms. Their purpose is to show syntactical relations, as well as the structure of the target items, over trees such as this one :



Here, all trees are considered ordered from left to right, in such a fashion that the trees

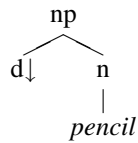


would be different.

Formally, a TAG is defined¹ as a set of *elementary trees*, containing :

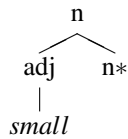
Initial trees: An initial tree has its internal nodes labeled with *non-terminal symbols*². Leaves are labeled either by terminal symbols³ or by non-terminal symbols marked with a *substitution marker*, \downarrow .

The following tree defines a lexical entry, *pencil*, of the *noun phrase* (np) type:



Auxiliary trees: An auxiliary tree is defined like an initial tree, of which at least one of the leaves is labeled by the same non-terminal symbol as its root. One of those leaves is called the *foot node* of the tree, and particularized using a * marker.

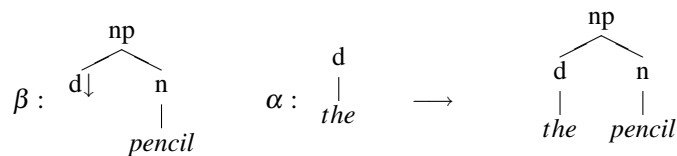
The following tree defines the adjective *small* as an optional complement to a *noun*:



Substitution and *adjunction* operations are used to get, from elementary trees, the *derived trees* of the grammar. They are defined as follows:

Substitution: A tree (initial or derived) α of root N can be substituted within a tree β , replacing a leaf labeled $N \downarrow$ in β by the tree α .

For instance, the following trees are combinable by substitution:



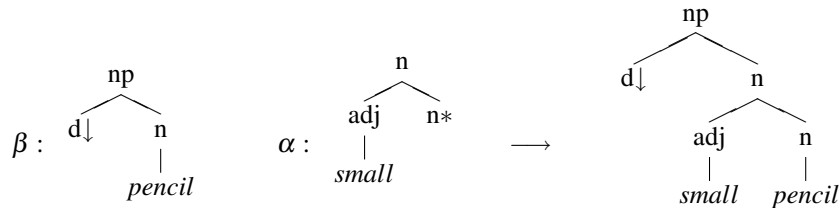
¹Those definitions and examples are from [Van93].

²modeling, usually, grammatical or syntactical classes, such as N , V , or np , vp ...

³Usually words or phrases in the sentence.

Adjunction: An auxiliary tree α whose root (and therefore whose foot node) is labeled N can be adjoined to a tree β , at a node n of β labeled N . That node is replaced by α , and the former subtree β with n as its root is then rooted at the leaf that was α 's foot node.

For instance, the following trees are combinable by adjunction:



Consider a given TAG G , with S as a start symbol. The set of trees, of root S , that can be derived from its elementary trees, and that only have terminal symbols on their leaves is called the *tree language* of G . Its *string language* is the set of strings formed by concatenation of those terminals, read on the usual order.

TAGs are mildly context-sensitive formalisms, and allow for a parsing $O(n^6)$ times for a sentence of length n .

An application of TAGs is the *lexicalization* of context-free grammars, i.e., the ability to model the CFG rules with at most one terminal symbol associated to one rule for any given CFG, without modifying the syntactical structure it models. Without TAGs, the lexicalization of a CFG (its transformation into its *Greibach normal form*) is destructive with respect to the derivation tree – see [JS97] for discussion. A lexicalized grammar is useful in order to create a dictionary with the terminal phrase associated to a rule as key, for instance. . .

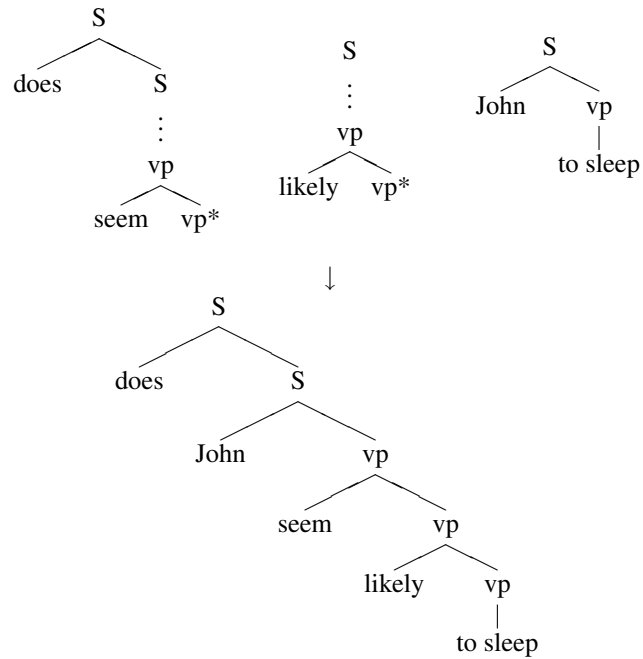
3.1.1 Multiple component tree adjoining grammars

Multiple Component Tree Adjoining Grammars (MCTAGs) is a term that comprises several extensions to TAGs – see [Kal05] for more precise definitions and examples. MCTAGs have simultaneous adjunctions occurring at the same step. As many variations exist, we will consider set-local MCTAGs that forces all adjunctions occurring at a given step to apply to a certain, defined set of elementary trees.

This change greatly increases the class of generated languages, as is proven in [Har01b]: the resulting formalism share its language class with MGs and MFCGs, among others⁴.

⁴Some MCTAG variants have a different generating power, which is sometimes not clearly characterized. See [SCD00] for details.

An example taken from [SCD00] can illustrate how MCTAGs can help define elaborate grammatical constructions such as "Does John seem likely to sleep ?":



Co-occurring adjunctions on nodes labeled *S* and *vp* allow the correct derivation to take place.

3.2 Generalized context-free grammars and their restrictions

[SMFK91] covers in detail several formalisms introduced earlier (GCFGs, PMCFGs, MCFGs and LCFRSs), that we shall sum up in this section. They generalize the concept of *context-free grammars*, which contains generative grammars with rules of the form:

$$A \rightarrow \alpha$$

where A is a non-terminal symbol and α is a string of non-terminal and/or terminal symbols (e.g., letters).

Generalized Context-Free Grammars (GCFGs), the first of these formalisms, uses analogous rules:

$$\begin{aligned} A_0 &\rightarrow f[A_1, \dots, A_q] \\ A_i &\rightarrow \alpha_i \end{aligned}$$

where each A_i is a non-terminal symbol, each α_i is a string of terminal symbols and f is a function.

The properties of these grammars, detailed thereafter, allow the expression of type-0 languages, and make them too general.

The introduction of constraints over the rules and functions is used to define Parallel Multiple Context-Free Grammars (PMCFGs), Multiple Context-Free Grammars (MCFGs), and Linear Context-Free Rewriting Systems (LCFRSs).

3.2.1 Generalized context-free grammars

A GCFG G is defined by:

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

where N is a finite set of non-terminal symbols (A_0, A_1, \dots), including the start symbol S . The items in O are *tuples* of strings over a set of terminal symbols T (i.e., $\alpha \in O \Rightarrow \exists k / \alpha \in (T^*)^k$), those in F are partial functions f in $O^q \rightarrow O$ (for a given integer q), and the rules in P are of the form:

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

There can be more than one instance of a given non-terminal symbol A_i in the right-hand side. When $q = 0$, the rule is a terminal one ($A \rightarrow f[]$ and $f[] = \alpha$ amounts to $A \rightarrow \alpha$, for a tuple of strings of terminal symbols $\alpha \in O$).

The string language of G for a given non-terminal A , $L_G(A)$, is defined as the smallest set of strings of terminal symbols such that:

- If $A \rightarrow \theta \in P$, where $\theta \in O$, (the rule is terminal), then $\theta \in L_G(A)$.
- If it is true that:
 - $\theta_i \in L_G(A_i)$ for $1 \leq i \leq q$,
 - $A \rightarrow f[A_1, \dots, A_q]$ is a rule in P , et
 - $f[\theta_1, \dots, \theta_q]$ is defined,
 then $f[\theta_1, \dots, \theta_q] \in L_G(A)$.

One can then define $L_G = L_G(S)$ the string language generated by G .

3.2.2 Parallel k–multiple context-free grammars

Parallel k–multiple context-free grammars are GCFGs over which some structural properties have been imposed.

A GCFG $G = (N, O, F, P, S)$ is a m –PMCFG if it satisfies:

- $O = \bigcup_{i=1}^m (T^*)^i$, where T is a set of terminal symbols distinct from N . In other words, O is the set of all tuples of strings of symbols in T , that are of arity m and less.
- Let $a(f)$ be the arity $f \in F$. Let also $r(f)$ and $d_i(f)$ for $1 \leq i \leq a(f)$ be positive integers lower than m , such that f is a function of:

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

Thus, every argument of f is a tuple of fixed, known arity, and the same holds for its result on the right-hand side, of strings of symbols from T .

In addition, each component of the right-hand side of f must be a concatenation (not necessarily linear) of given constant strings of t and of items from the arguments of f , e.g., with z_i as left-hand side variables and α, β constant strings:

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

- For every $A \in N$, let $d(A)$ be the *dimension* of the non-terminal symbol A . The following condition is then imposed upon the used functions: if $A \rightarrow f[A_1, \dots, A_{a(f)}] \in P$, then $r(f) = d(A)$ and, for $1 \leq i \leq a(f)$, $d_i(f) = d(A_i)$. The non-terminal symbols are therefore of fixed arity (dimension), which is equal to that of the arguments of the functions that may be applied to them.
- With these conventions, $d(S) = 1$.

k -PMCFGs therefore impose an arity to each their non-terminal symbols. The maximal arity of these symbols is k , and it determinates, to some extent, the generative power of the grammar:

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

One can thus define, using a 1-PMCFG, the language $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$:

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

Here, A is of dimension $d(A) = 1$ (A can only be derived into items from $T : 0$ or 1), g_1 is a function on rank 2 (as it takes two arguments, each of dimension 1), and g_2 is of rank 1. As the maximal dimension of used symbols is 1, it is indeed a 1-PMCFG, and $O = \bigcup_{i=1}^1 (T^*) = T^*$.

A derivation sequence generating a word of the language using that grammar could be, for instance:

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

Using another 1-PMCFG, the language $\{\alpha^{(2^n)}\}$ can also be defined (here, using $\alpha = a$):

$$\begin{aligned} T &= \{a\}, N = \{S\}, O = T^* \\ S &\rightarrow a \mid g_1[S] \\ g_1[x] &= xx \end{aligned} \quad 1$$

The language $\{a^{(n^2)} \mid n > 0\}$ can be defined, too, with the following 2-PMCFG:

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

In that case, the symbol A is of dimension 2, and the functions used are of rank 1 but operate over string pairs. A derivation sequence could be:

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

3.2.3 k-multiple context-free grammars

A k -multiple context-free grammar (k -MCFG) is a k -PMCFG upon which the *non-copy condition* has been enforced. This condition is true when a variable of any given function f (i.e., one of the components of an argument of f 's left-hand side) to be used more than once in the right-hand side of f .

The language $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$ (already defined using a 1-MCFG) can be redefined, using a 2-MCFG:

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

For any given m , the language $\{a_1^n a_2^n \dots a_m^n \mid n \geq 0\}$ can be generated by a m -MCFG using the following pattern:

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

3.2.4 Linear context-free rewriting systems

Though introduced independently from GCFGs, PMCFGs and MCFGs, linear context-free rewriting systems (LCFRSs) are also a restriction of this formalisms. Thus, a k -LCFRS is a k -MCFG verifying the *non-erasure condition*, that is to say that a given variable of a function f must be used exactly once (not one time or more) in the right-hand side of f .

This particular restriction has no influence over the generative power of the formalism: $L(\text{LCFRS}) = L(\text{MCFG})$.

Thus, the 2-MCFG covered above, generating $\{\alpha^2 \mid \alpha \in \{0, 1\}^+\}$, satisfies this condition and is also a 2-LCFRS:

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

3.3 Range concatenation grammars

A range concatenation grammar (RCG), as defined in [Bou98], is a set of *rules* made of *predicates* over *ranges* of strings.

Let $N = \{X, Y, Z, \dots\}$ be a finite set of non-terminal symbols and $O = \{a, b, c, \dots\}$ be a finite set of terminal symbols. The rules of a RCG are given thus:

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

where the ψ_i are *predicates* of a given arity p , and:

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

with $\alpha_i \in (O \cup N)^*$. Let S be singled out as the *start predicate*.

Non-terminal symbols may not be used more than once for a given predicate. *Negative* predicates can also be used, such as $\overline{A(X)}$ which is true whenever $A(X)$ is false.

RCG operate over *ranges* of the sentence.

A *range* is a substring of words of the language, and is formally defined, given a string $w = a_1 \dots a_n$, as a pair (i, j) , $0 \leq i \leq j \leq n$.

(i, j) represents then the substring of w reading $a_{i+1} \dots a_j$. The range (i, j) is of length $j - i$, and is empty whenever $i = j$ ($(i, j) = \varepsilon$). Consecutive, coherent ranges can be concatenated, and then $(i, j) \bullet (j, k) = (i, k)$.

The arguments of predicates in a RCG rule can be bound to the ranges of a given string w , in a coherent way regarding concatenation: $A(aXb)$ can, for instance, be bound to any string beginning with a and finishing with b .

For instance, $A(aX, bY) \rightarrow B(X, Y)$ can be instantiated to $A((i, j), (k, l)) \rightarrow B((i + 1, j), (k + 1, l))$, if the string w is such that $a_{i+1} = a$ and $a_{k+1} = b$.

A string w is generated by an RCG G when the empty range ε can be derived from any valid instantiation of the start predicate S , using G 's rules. The string language of G is the set of these strings.

The following example RCG, from [Bou99], defines the language $\{w^3, w \in \{a, b\}^*\}$:

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

The following sequence is a possible derivation of *abaabaaba*:

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

The language $\{a^n b^n c^n d^n, n > 0\}$ can also be defined, using the rules:

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

In that case, *aabbccdd* would be recognized using this derivation sequence:

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

3.3.1 Simple positive range concatenation grammars

A RCG is called *positive* when it does not contain any *negative* predicate (of the $\overline{A(X)}$ kind). This property does not alter the generative power of general RCGs:

$$L(\text{PRCG}) = L(\text{RCG})$$

A RCG G is called *simple* when it satisfies a certain set of properties (analogous to the properties of LCFRS), i.e., when G is:

Non-combinatory: A given rule $A_0(\dots) \rightarrow A_1(\dots) \dots A_k(\dots)$ cannot use terminal symbols in its right-hand side predicates, and only defines the ranges of $A_1 \dots A_k$ using only the non-terminal symbols used on the left-hand side for the ranges of A_0 . (In other words, rules such as $A(X) \rightarrow B(aX)$ may not be used.)

Linear: A rule cannot use more than once the same non-terminal symbol in the right-hand side: $A(X) \rightarrow B(X) C(X)$ may not be used.

Non-erasing: A rule must use every non-terminal symbol used on the left-hand side, in the right-hand side: $A(X, Y) \rightarrow B(X)$ may not be used.

Simple positive range concatenation grammars (sPRCGs) are RCGs that verify both conditions.

The generated language class is equal to that of MCFGs, this formalism being, esthetic considerations aside, identical to LCFRS. (Those results are detailed in [Bou98] and covered here in section 4.1.2, p. 27.)

3.4 Minimalist grammars

Minimalist grammars (MG), defined in [Sta97b] by Edward Stabler, are a formal adaptation of Noam Chomsky's *minimalist program*. This *minimalist program* represents, in a purified, essential sort of way, the mechanisms which to be shared by all human languages, and would therefore be part of an hypothetical *universal grammar*. Such a grammar would supposedly be innate to the human being and could, by adjusting some parameters, be turned into any human language. Whatever the truth of those hypotheses, the minimalist program, by virtue of its very construction, can model all phenomena studied within that school of linguistics (of which there are many).

Its principles are founded over the notion of *movement*. Thus, minimalist grammars can easily take a change to the utterance's form⁵ into account.

⁵The interrogative form of an affirmative sentence in English, for instance, implies a movement of the subject or complement around the verb.

3.4.1 Features and the lexicon

Minimalist grammars are entirely defined with *lexical expressions*, which in their turn are made of *features* representing certain properties. These features can be:

Phonological features: modeling the word, word part or word phrase phonetically (representing its sound) or as it is written (representing its letter), depending on the needed output. These features are those present in a given sentence, whether generated by the grammar or meant to be parsed.

Semantical features: modeling the sense of the associated phonological features. These can be ignored as a simplification, in the case, as it is here, of a formalism that won't be used for a semantical analysis, but they are included in the specification of the grammars.

Syntactical features: modeling the structure that will be constructed by derivation, that should represent the syntactical construct of the sentence.

The following feature sets are also defined:

Non-syntactical features: The set V of the non-syntactical features, i.e., of phonological and semantical features.

Base categories: The set B of the *base categories*, subset of the syntactical features. This set contains the syntactical classes of the sentence, i.e., in English v for a verb, n for a noun, and so on. At least one *completeness* (or *start*) *category* $c \in B$ is singled out⁶ and will play a role akin to that of start symbols S in generative grammar.

Selectors: The set S of the *selectors*, noted ' $= D$ ' with $D \in B$. These features represent a demand, or need, of a base category in a given place.

Licensors: The set $+L$ of *licensors*, $+f$. These features represent a *property*, that can induce a move. In human languages, these features are used for representing *cases* (nominative, ablative...) that are associated to grammatical functions, or to indicate the nature of a structure (interrogative, ...), for instance.

Licensees: The set $-L$ of the *licensees*, $-f$. Symmetrically to the licensors, these features indicate a demand of a property. A licensee $-f$ corresponds to the need of a property expressed by the licensor $+f$; in the case of human languages, the case is required by the structure with the feature $-f$ and provided by an expression containing the feature $+f$.

⁶ c , or C , is for *complementizer* and designs a syntactical structure, possibly empty, that can be added to a complete sentence. This optional complementizer does not change the character of the sentence, that is part of the generated language. Exemple of complementizers include the English *That*, ..., the Japanese *Nee*, ..., the French *Que ...!*, and the empty word. We will follow Jens Michaelis and call c the *completeness category* throughout this study, though several other terms have been designed.

Finally, let us define an *elementary expression* of a minimalist grammar as a string of non-syntactical features, e.g., $t \in V^*$ and the corresponding sequence of syntactical features, e.g., $\gamma \in (B \cup S \cup +L \cup -L)$. There are two kinds of elementary expressions that can be constructed: *lexical entries* defined in the grammar's lexicon, denoted with a $\{::\}$ marker, e.g., $t :: \gamma$, and *derived expressions* constructed by a derivation process, denoted by a $\{:\}$ marker, e.g., $s : \gamma'$.

As we shall see, expressions can be *complex*, i.e. composed of a sequence of elementary expressions (separated by commas), such as $u_1 : \gamma_1, u_2 : \gamma_2, \dots$

The lexicon of the grammar is then a set of elementary expressions such as:

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

The canon use is to restrict the forming of the lexical entries as follows:

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

However, numerous variations exist (of note, some grammars use more than one licenser in their entries). These forming variations do not in themselves change the formalism.

3.4.2 Simple minimalist grammars

The following definition is not the one traditionally used. As we are primarily concerned with string languages, we define minimalist grammars as operating over sequences of strings. It also doesn't use many functionalities found in Edward Stabler's definition (see the next section for details). But the formalism defined here has all the properties needed, and can express any "traditional" minimalist grammar.

A minimalist grammar is completely defined by its lexicon. The expressions are derived using *merge* and *move* operations.

An elementary expression that contains only the (or one of the) *completeness categorie(s)* of the grammar in its syntactical features is terminal, and its non-syntactical features belong to the generated language.

Let us consider:

- s, t and some $u_i, v_i \in V^*$, some non-syntactical feature strings (typically, words);
- $f \in B$ a given base category;
- $=f \in S$ the selector associated to f ;

- γ and some $\alpha_i, \beta_i \in (B \cup S \cup +L \cup -L)^*$ some (possibly empty) strings of syntactical features; and
- $\delta \in (B \cup S \cup +L \cup -L)^+$ a non-empty string of syntactical features.

We will use a single dot $\cdot \in \{:, ::\}$ to represent either the lexical ($\{:\}$) or the derived ($\{::\}$) expression marker.

Thus, an expression might be written as $t : f, u_1 : \alpha_1, \dots, u_k : \alpha_k$, and will be *complex* if $k > 0$ (composed from $k + 1$ elementary expressions) and *elementary* otherwise, *derived*, with its *head* (foremost elementary expression) being composed of the non-syntactical feature string t in association of the single base feature f .

Derivation operations take either one or two expressions A, B to produce another expression C , which we note $\frac{A \ B}{C}$ or $\frac{A}{C}$. They can be defined as:

Merge: A *merge* of two expressions corresponds to the fulfillment of a selection need, i.e., the integration, by an expression including a selector $= D$, of an expression including the base feature D .

This operation varies according to the situations:

Selection of a simple-headed expression by a lexical expression: The lexical expression $s :: = f\gamma$, is combined with the expression $t \cdot f, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k$ (of which the *head*, i.e., the first component, is reduced to the selected base feature). The following derived expression is then formed:

$$\frac{s :: = f\gamma \quad t \cdot f, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k}{st : \gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k}$$

Selection of a simple-headed expression by a derived expression: The derived expression $s := f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k$, is combined with the expression $t \cdot f, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l$ (of which the *head*, once again, is reduced to the selected base feature). The following derived expression is then formed (in an analogous way to the former derivation, but the non-syntactical features are reversed):

$$\frac{s := f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k \quad t \cdot f, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}{ts : \gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}$$

Selection of a complex-headed expression: The (lexical or derived) expression

$s \cdot = f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k$ is combined with the expression $t \cdot f\delta, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l$ (here, by contrast, the head contains the selected base feature plus one or more syntactic features, that can cause a merge or move – recall that δ is non-empty). The following derived expression is then formed:

$$\frac{s \cdot = f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k \quad t \cdot f\delta, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}{s : \gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k, t : \delta, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}$$

Move: The *move* in a composite expression is the realization of a licensing, i.e., the association of a licensee $-f$ and of the corresponding $+f$ licenser. Two cases are distinguished:

Final licensee move: The derived expression

$s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k$
(the head contains at least the $+f$ feature, while one of the subsequent elementary expressions $u_i \cdot \alpha_i$ is *reduced*, in its syntactical features, to $-k$), is applied a move. The following expression is then formed:

$$\frac{s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}{ts : \gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}$$

Non-final licensee move: The derived expression

$s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f\delta, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k$
(the head contains at least the $+f$ features while one of the subsequent simple expressions $u_i \cdot \alpha_i$ *starts with* $-k$ as a syntactical feature *but is not reduced to it*), is applied a move. The following expression is then formed:

$$\frac{s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f\delta, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}{s : \gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : \delta, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}$$

3.4.3 General minimalist grammars

The minimalist grammar definition given above is not exactly the one from [Sta97b].

Indeed, the original article uses additional notions, notably of *strong* ($+X$) and *weak* ($-x$) features, allowing to apply either a move on the whole expression, as defined before, or a move leaving the phonological features untouched, called *covert phrasal move*, as well as a *head movement* operation (a specific case of selection / merge).

However, [Mic01a] proves that these notions do not affect the class of string languages.

Other functions such as adjunction have since then been added to the minimalist grammars, forming as many variations, called here *general (or complete)* minimalist grammars, or hMG.

We shall not cover in detail the modifications to be used for those functions, but it is a matter, in general, of adding the relevant cases to the appropriate merge and move operations. Those variations, even if they do not change the language class of the formalism, give a more straightforward way to express some linguistic phenomena.

It is also possible to extend minimalist grammars beyond the generative power of the MG and hMG studied here : [MK05] thus introduces two variations of type-0 power (well beyond any mildly context-sensitive language...).

Chapter 4

Equivalences and conversion of the formalisms

In this chapter, we shall present some of the equivalences known to exist between formalisms.

Two formalisms are said to be *weakly* equivalent when their string language classes are the same, and *strongly* equivalent if, for any string language that can be expressed in both formalisms, the derivation processes are identical (i.e., the derivation trees are the same).

Here, unless expressly stated, the mentioned equivalences are *weak*.

4.1 Variations of PMCFGs, MCFGs and RCGs

4.1.1 PMCFGs and RCGs

In [Lju05], Peter Ljunglöf extends PMCFGs with an *intersection* operation. He goes on to prove that the formalism thus constructed is equivalent to Pierre Boullier's RCGs, and therefore that they, too, cover exactly the class of polynomial-time parsable languages, *PTIME*.

Though it bears little impact over the formalisms that concerns us most, [Lju05] is an important step towards the unification of *PTIME*-class formalisms.

4.1.2 LCFRSs and sPRCGs

[Bou98] points out that LCFRSs and sPRCGs are strongly (as well as weakly) equivalent. Indeed, the two formalisms, though they are defined from very different viewpoints, are identical aside from the writing. A LCFRS rule of the kind:

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

where the \vec{x}_i and \vec{t} are variables representing tuples of strings (and \vec{t} is a combination on components of the \vec{x}_i), is trivially equivalent to a sPRCG rule of the kind:

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

Predicates have here the role of non-terminal symbols in the LCFRS.

4.1.3 MCFGs in normal form

[SMFK91] introduces the notion of *normal form* for MCFG. It is demonstrated therein that, for any given MCFG G , there exists an MCFG G' weakly equivalent to G , such that:

- Any variable of a function in G' appears exactly once in the right-hand side of the rule (in other words, G' is a LCFRS).
- G' is proper: none of its rules use constant strings, apart from terminal rules.

Moreover, [Mic01a] points out that, without loss of generality, the functions of MCFGs can be considered as taking at most two arguments (when it is not the case, one just need to split a rule in two or more and repeat as needed).

Thus, [Gui04] defines and uses a *binary normal form* of MCFG, combining the aforementioned conditions. For any MCFG, there then exists a weakly equivalent binary normal form MCFG. The rules of a binary normal form MCFG are, therefore, of one of the following patterns:

- $A \rightarrow \alpha$, for a terminal rule (the non-terminal symbol A of arity k ($k = 1|2$) is rewritten into a constant k -uple of strings of terminal symbols α).
- $A \rightarrow f(B)$, where A, B are non-terminal symbols and f is a function that re-arranges the components of B . For instance, if A and B are both of dimension 2, we could have $f[(a, b)] = (b, a)$. In that case, in order to write the rule in simpler fashion, we could write

$$A \rightarrow B [0, 1][0, 0]$$

Here, the coordinates appended at the end of the rule modelize the arrangements of the variables used : $[0, 1]$ represents the second (number 1) component of the first (and only, number 0) element on the right-hand side, B , and so on. This way of writing MCFG rules, also introduced in [Gui04], clearly show how their implementation would be done.

- $A \rightarrow g(B, C)$, where A, B, C are non-terminal symbols and g maps the components of both B and C into the components of A . In a similar fashion, one can write the rules of that kind in a compact way, as in the following example:

$$A \rightarrow B C [0, 1; 1, 0][0, 0][1, 1]$$

Assuming that A is of dimension 3 and B and C are of dimension 2, the coordinates $[i, j]$ represent the variables in B and C to extract, and this formulation is equivalent to:

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

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

4.2 MG-MCFG equivalence and minimalist grammars parsing

[Mic01a] proves that, for any given minimalist grammar, there exists a MCFG generating the same language. [Mic01d] symmetrically proves that, for any given LCFRS, there exists a minimalist grammar generating the same language, and therefore, as LCFRSs and MCFGs are weakly equivalent, that the three formalisms share the same language class.

Moreover, [Mic01a] gives the formal construction of an MCFG corresponding to a given minimalist grammar. This is the process that concerns us in this section, and it is detailed and implemented in an efficient way in [Gui04].

4.2.1 The procedure of converting minimalist grammars into MCFGs

The conversion of a minimalist grammar G_{MG} into a MCFG G_{MCFG} consists of the representation, using non-terminal symbols of the MCFG, the entire structure of the minimalist grammar, regarding the possible merge and move operations.

[Mic01a] points out that, since that the lexicon of a minimalist grammar is finite, the set of derived expressions is finite as well.

Indeed, each and every merge or move operation erases two of the features of the expressions used (base and selector, or licensor and licensee). In addition, expressions can only be constituted of suffixes of the lexical expressions¹.

Associating every relevant (i.e., actually used) expressions, whether lexical or derived, with a non-terminal MCFG symbol, one models the derivation operations that lead to the constitution of these expressions with rules using the associated non-terminal symbols : one in the case of a move, two in the case of a merge.

Thus, the constructed MCFG will be written in normal binary form.

¹This is the main idea behind the detailed proof in [Mic01a]. Jens Michaelis elaborates over the closure of *relevant expressions*, i.e., the expressions that can be eventually derived into a terminal state, and their properties, in order to account for the finiteness of this set. This argument is also used for some sort of study in complexity, section 4.2.3, p. 39.

[Gui04] explains this procedure, that we will summarize herein. We shall use the notations previously used for the minimalist grammar rules, importantly $\cdot \in \{:, ::\}$ will be used to represent the kind of (elementary) expression considered ($\{::\}$ for a lexical and $\{:\}$ for a derived one). We need to study each possible merge (three cases) and move (two cases) operation, as detailed section 3.4, p. 21.

1. Conventionally, for every sequence of syntactical features e (used in an elementary or complex expression) is associated a non-terminal symbol in the MCFG, $\sigma(e)$. These symbols are noted t_0, t_1, \dots and $t_7 = \sigma(= D = D C)$ simply means that the non-terminal symbol t_7 will play, in the MCFG, the role of the sequence of syntactical features $= D = D C$. Afterwards, we will consider every possible MG derivation rule.
2. When the following derivation rule is used in the MG:

$$\frac{s :: = f\gamma \quad t \cdot f, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k}{st : \gamma, \alpha_1, \dots, \alpha_k}$$

The non-terminal symbol $\sigma(\gamma, \alpha_1, \dots, \alpha_k)$, is added to the MCFG, as well as the rule:

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

Also written:

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

3. When the following derivation rule is used in the MG:

$$\frac{s : = f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k \quad t \cdot f, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}{ts : \gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}$$

The non-terminal symbol $\sigma(\gamma, \alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_l)$, is added to the MCFG, as well as the rule:

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

Also written:

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

4. When the following derivation rule is used in the MG:

$$\frac{s \cdot =f\gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k \quad t \cdot f\delta, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}{s : \gamma, u_1 \cdot \alpha_1, \dots, u_k \cdot \alpha_k, t : \delta, v_1 \cdot \beta_1, \dots, v_l \cdot \beta_l}$$

The non-terminal symbol $\sigma(\gamma, \alpha_1, \dots, \alpha_k, \delta, \beta_1, \dots, \beta_l)$, is added to the MCFG, as well as the rule:

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

Also written:

$$\sigma(\gamma, \alpha_1, \dots, \alpha_k, \delta, \beta_1, \dots, \beta_l) \rightarrow g[\sigma(=f\gamma, \alpha_1, \dots, \alpha_k), \sigma(f\delta, \beta_1, \dots, \beta_l)] \\ g[(x_0, x_1, \dots, x_k), (y_0, y_1, \dots, y_l)] = (x_0, x_1, \dots, x_k, y_0, y_1, \dots, y_l)$$

5. When the following derivation rule is used in the MG:

$$\frac{s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}{ts : \gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}$$

The non-terminal symbol $\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k)$, is added to the MCFG, as well as the rule:

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k) \rightarrow \sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f, \alpha_{i+1}, \dots, \alpha_k) \\ [0, i; 0, 0] [0, 1] \dots [0, i-1] [0, i+1] \dots [0, k]$$

Also written:

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k) \rightarrow g[\sigma(+f\gamma, \alpha_1, \dots, \alpha_k)] \\ g[(x_0, x_1, \dots, x_k)] = (x_i x_0, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$$

6. When the following derivation rule is used in the MG:

$$\frac{s : +f\gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : -f\delta, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}{s : \gamma, u_1 \cdot \alpha_1, \dots, u_{i-1} \cdot \alpha_{i-1}, t : \delta, u_{i+1} \cdot \alpha_{i+1}, \dots, u_k \cdot \alpha_k}$$

The non-terminal symbol $\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k)$, is added to the MCFG, as well as the rule:

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k) \rightarrow \sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, +f\delta, \alpha_{i+1}, \dots, \alpha_k) \\ [0, 0] [0, 1] \dots [0, k]$$

Also written:

$$\sigma(\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k) \rightarrow g[\sigma(+f\gamma, \alpha_1, \dots, \alpha_{i-1}, +f\delta, \alpha_{i+1}, \dots, \alpha_k)] \\ g[(x_0, x_1, \dots, x_k)] = (x_0, x_1, \dots, x_k)$$

7. The starting point of the algorithm is to represent the lexicon of the minimalist grammar as MCFG rules. Thus, for any lexical expression $s :: \delta$, the following terminal rule is added to the MCFG: $\sigma(\delta) \rightarrow s$.
8. To conclude the conversion, we need to represent, as an initial MCFG rule, the possible terminations of the derivations of the minimalist grammar. For any completeness category C defined in the MG, this rule is added : $S \rightarrow \sigma(C)[0, 0]$ (where S is the start symbol of the MCFG) – in usual notation:

$$S \rightarrow f[\sigma(C)] \\ f[x] = x$$

Intuitively, the MCFG encodes the syntactical features of the minimalist grammars using rules, while the non-syntactical features of each expression is conveyed via the non-terminal symbols. The resulting MCFG takes therefore into account exactly all the possible combinations of the lexical expressions intrinsic to the original minimalist grammar.

By construction, the derivation steps during the parsing or generation of a sentence are exactly equivalent. However, MCFGs do not directly provide the structural tree of a sentence, unlike MGs (see Appendix A, p. 45).

On another hand, [Gui04] also proposes procedures with some minimalist grammar variations (hMG) in their scope, to associate every derivation with a MCFG rule, in a similar way to the conversion detailed herein.

4.2.2 Conversion example

We will use here an example of grammar generating the counter languages of the kind $\{a_1^n a_2^n \dots a_k^n\}$, the implementations of which have been realized by Maxime Amblard (using the lexical formatting allowing for multiple licensors), in [Amb05].

The procedure is identical for every grammar of this kind, we will detail the case of $\{a^n b^n\}$, tested in that form with Matthieu Guillaumin's program. (As the conversion is a lengthy process to describe in full, we use this relatively simple language, but the observations also hold for, e.g., $\{a^n b^n c^n\}$.) We will present the MCFG rules using the binary normal notation introduced earlier, for the sake of simplicity.

The minimalist grammar G_{MG} , whose base categories are $\{D, V, C\}$ and the accepting symbol is C , generating the language $\{a^n b^n\}$, is defined by the following lexicon:

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

To construct the equivalent MCFG G_{MCFG} , let us first add the non-terminal symbols t_0 to t_5 , corresponding to the following feature sequences (directly extracted from the lexical expressions):

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

The lexicon also directly provides the following terminal rules of G_{MCFG} :

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

To complete the initialization of the algorithm, let us add to G_{MCFG} the initial symbol S and the rule:

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

Then, let us apply the closure of possible derivations from the lexical expressions, adding at each step the adequate symbols and rules:

- The symbols t_0 and t_3 , modeling the expressions $D - D$ and $= D + V V - V$, can be combined in order to model a merge between $= D$ and D , forming an hypothetical t_6 symbol representing $+V V - V$, $-D$. However, this composite expression cannot be solved: no possible derivation can erase the first symbol of its head, $+V$ (in fact, the $-V$ features is not present in any other component of this expression, preventing all movement). In that case, the algorithm forsakes this derivation branch and adds neither symbol nor rule to G_{MCFG} .
- The symbols t_0 and t_2 , modeling the expressions $D - D$ and $= D V - V$, can be combined in order to model a merge between $= D$ and D , forming a t_7 symbol representing the expression $V - V$, $-D$. Here, by contrast to the previous case, a rule can be applied to the resulting symbol, using a merge with another expression, the first head feature of which would be $= D$ (expression already existing in the lexicon). The t_7 symbol is added to G_{MCFG} , as well as the rule:

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

- The symbols t_5 and t_7 , modeling the expressions $= V + D + V C$ and $V - V$, $-D$, can be combined in order to model a merge between $= V$ and V , forming a t_8 symbol representing the expression $+D + V C$, $-V$, $-D$. Here too, a move rule can be applied to the new expression, upon two of its components: its head beginning with $+D$ and another, beginning with $-D$. The t_8 symbol is added to G_{MCFG} , as well as the rule:

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

- The symbols t_1 and t_7 , modeling the expressions $= V + D D - D$ and $V - V$, $-D$, can be combined in order to model a merge between $= V$ and V , forming a t_9 symbol representing the expression $+D D - D$, $-V$, $-D$. The t_9 symbol is added to G_{MCFG} , as well as the rule:

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

- The t_9 symbol, modeling the expression $+D D -D, -V, -D$, can be modified in order to model a move from $+D$ to $-D$, forming a t_{10} symbol representing the expression $D -D, -V$. The t_{10} symbol is added to G_{MCFG} , as well as the rule:

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

- The t_8 symbol, modeling the expression $+D +V C, -V, -D$, can be modified in order to model a move from $+D$ to $-D$, forming a t_{11} symbol representing the expression $+V C, -V$. The t_{11} symbol is added to G_{MCFG} , as well as the rule:

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

- The t_{11} symbol, modeling the expression $+V C, -V$, can be modified in order to model a move from $+V$ to $-V$, forming a t_{12} symbol representing the derived expression reduced to C . The t_{12} symbol is added to G_{MCFG} , as well as the rule:

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

- The symbols t_3 and t_{10} , modeling the expressions $= D +V V -V$ and $D -D, -V$, can be combined in order to model a merge between $= D$ and D , forming a t_{13} symbol representing the expression $+V V -V, -D, -V$. The t_{13} symbol is added to G_{MCFG} , as well as the rule:

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

- The t_{13} symbol, modeling the expression $+V V -V, -D, -V$, can be modified in order to model a move from $+V$ to $-V$. The expression formed by this derivation is $V -V, -D$, corresponding to the t_7 symbol. Therefore, only the following rule is added to G_{MCFG} :

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

There is no other possible derivation of those expressions. To finish, the following rule, corresponding to the t_{12} symbol, created in the previous steps along with the $t_{12} \rightarrow t_{11} [0, 1; 0, 0]$ rule and which is associated to the completeness category C , is added to G_{MCFG} :

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

In the end, the resulting MCFG G_{MCFG} contains the following non-terminal symbols:

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

The rules of G_{MCFG} are as follows:

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

4.2.3 Conversion complexity

Edward Stabler pointed out that, using the conversion procedure as it is implemented here, complexity issues could defeat the prospect of an efficient conversion: for some minimalist grammars, the converted MCFG could be exponentially larger than the source grammar. That would be the case, in particular, with Jens Michaelis' grammars for counter languages ($\{a_1^n a_2^n \dots a_k^n\}$ for an increasing k), for which the conversion into MCFG very quickly grows, even as there exist simpler MCFGs for the same languages.

Our hypothesis is that, the conversion using only the intrinsic syntactical structure given by the minimalist grammar, the size of the result MCFG, and therefore the complexity of the conversion, depends only on the way of writing the source minimalist grammar, and that the fact that the result MCFG is exponential signifies only that the MG is not efficient. In particular, as the conversion represents exactly the possible derivation steps induced by the lexicon of the MG, we conjecture that the conversion cannot be of a significantly greater complexity than solving the general problem of the parsing of any given sentence using the original minimalist grammar – indeed, we would surmise these two problems to be equivalent.

This hypothesis has been reinforced by the fact that, for the counter languages described earlier, the conversion realized over some alternative minimalist grammars, these written by Maxime Amblard in [Amb05] and used in the previous section, gives very good results: for a source grammar of size $2n + O(1)$, the result MCFG is of size $3n + O(1)$.

A problem of that approach is that it does not appear immediately that a particular minimalist grammar would be "better" than another, for a given language. We do not have, in particular, any algorithm able to demonstrate whether a grammar is the most efficient for a given language...

We then tried to bound the size complexity of the resulting MCFG. It appears that a minimalist grammar can indeed generate a MCFG of great size, under certain conditions; the most important parameter is the maximal length of a lexical expression, and mostly the maximal numbers of selectors and licensors in an expression. We can give two simple arguments to that end, considering a grammar of arbitrarily numerous derivation possibilities.

Time analysis – based on conversion algorithm steps

First, let us consider a minimalist grammar containing n lexical expressions, each containing at most s selectors and m licensors:

$$\left\{ \begin{array}{l} u_1 :: = f_1 \dots = f_{sel_1} - l_1 \dots - l_{lic_1} \dots \\ \dots \\ u_i :: = f_i \dots = f_{sel_i} - l_i \dots - l_{lic_i} \dots \\ \dots \\ u_n :: = f_n \dots = f_{sel_n} - l_n \dots - l_{lic_n} \dots \end{array} \right. , s = \max_{1 \leq i \leq n}(sel_i), m = \max_{1 \leq i \leq n}(lic_i)$$

The algorithm is executed in *steps*, trying, at every iteration, to use every possible derivations for all the expressions that have already been obtained. Let n_i note the maximal number of expressions produced during step i :

$$\begin{array}{l}
 (\text{step } 0 : \text{lexicon}) \\
 (\text{step } 1) \\
 \dots \\
 (\text{step } i)
 \end{array}
 \left\{ \begin{array}{l}
 u_1 :: \dots \\
 \dots \\
 u_n :: \dots \\
 v_{1,1} :: \dots \\
 \dots \\
 v_{1,n_1} :: \dots \\
 \dots \\
 v_{i,1} :: \dots \\
 \dots \\
 v_{i,n_i} :: \dots
 \end{array} \right.$$

We have $n_0 = n$, the first step integrating only lexical expressions, and $n_1 = n^2$, the only possible derivations at first being the combinations (by merge) of lexical expressions. Then, at any step i , the n_{i-1} expressions produced at the latest step can merge each with every other expressions previously produced.

The maximal number of expressions produced by merge at step i is therefore $g_i = n_{i-1} \sum_{k=0}^{i-2} (n_k)$.

At that same step i , every item produced at the latest step can also be applied a move. The maximal number of moves is therefore $v_i = n_{i-1}$.

We have $n_i = g_i + v_i$, i.e., $n_i = n_{i-1} (1 + \sum_{k=0}^{i-2} (n_k))$.

On the other hand, each lexical expression can, be applied s merges and m moves at most, the number of steps is therefore bound by $s + m$: there can be at most but $n + n^2 + \sum_{i=2}^{s+m} (n_i)$ ways to generate expressions in total.

This complexity can be reduced to $O(n^p)$, p being the greatest exponent of the sum – that of n_{s+m} .

Let us then examine E_i , the greatest exponent of n_i :

- $E_0 = 1$ (for $n_0 = n$),
- $E_1 = 2$,
- and $E_i = E_{i-1} + E_{i-2}$ (for the greatest exponent of $n_i = n_{i-1} (1 + \sum_{k=0}^{i-2} (n_k))$ is that of $(n_{i-1})(n_{i-2})$),
- and it can be noticed that $E_i = F_{i+2}$, where F is the usual Fibonacci sequence.

Therefore, we have:

Result 1 *The complexity in time of the conversion is, at worst, $O(n^{F_{s+m+2}})$.*

That quantity is, obviously, very strongly growing with $s + m$ (comparable to $O(n^{4^{s+m}})$).

Space analysis – based on maximal derivable expressions

Considering again a grammar containing n lexical expressions, each with a syntactical feature strings of length at most l and containing at most s selector features.

Notice that each derived expression contains at most 2^s components. Indeed, only the merge operation allows the adjunction of new components to an expression, equal in number to that of the adjoined expression.

The expression head being formed from a lexical expression, the latter can be applied s merges at most, in the course of which the component number is at most doubled.

Moreover, each derivation erasing the first symbol of one or more of the components, those can only be made of lexical expression suffixes.

However, there are at most l suffixes for every lexical expression, and hence nl suffixes all told.

There could therefore be $(nl)^{2^s}$ expressions, at most, that are derivable from the lexicon.

We therefore have :

Result 2 *The size complexity of the result MCFG is less than $O((nl)^{2^s})$.*

That quantity is (in general) lesser than the former result. However, as it does not take every step of the algorithm into account, the time and memory used for the conversion likely are of a greater complexity.

4.2.4 Parsing minimalist grammars

Since the formalization of minimalist grammars, several parsers for those formalisms have been implemented, for example one from Harkema, detailed in [Har01a]. These programs are, in general, variations of the Cocke-Younger-Kasami algorithm for context-free grammars (introduced in [Kas65], [You67], and [CS70]).

A practical part of the research of equivalent formalisms to minimalist grammars is to compare the performance of those parsers with a procedure involving the conversion of minimalist grammars to another formalism, and the subsequent appliance to the result of an already known and efficient parser for the end formalism.

Thus, there exists a MCFG parser introduced by Daniel Albro in [Alb00], that is not completely correct, as it only works for MCFGs written in normal binary form – however, as the conversion algorithm uses exactly that output, this is not a problem. This algorithm is a variation of Earley’s algorithm for context-free grammars (introduced in [Ear70]). The complexity of the parsing of a sentence of n words with an MCFG of c categories (maximal number of variables used in any rule, most of the time 2) is of $O(n^{4c+3} \log_2 n)$.

In [Gui04], Guillaumin benchmarks the efficiency of the different algorithms he has access to, as well as the procedure of conversion and parsing of the resulting MCFG, using a few test sentences. The results are without appeal, the parsing after conversion being faster than the best minimalist grammar parsing algorithm by one (0.2 second against about 5) to four (0.6 second against about 43 *minutes*) orders of magnitude.

The experimental results do seem to validate this method, even if the benchmarks realized were very restricted due to the small quantity of test cases available. Furthermore, the theoretical difference remain hard to assess: as seen earlier, depending on the characteristics of the minimalist grammar used, the conversion (and, most probably, the analysis using any implementation that might exist) can reach exponential complexity.

If minimalist grammars are indeed the formalism used unconsciously for the representation of human languages, however, one can assume that the grammars actually used would be either be processed in a more efficient way than our worst-case scenarii suggest, or that they are written in such a way that their analysis is simple.

In an illustration of such an hypothesis, in [Sta97a], Edward Stabler formalizes an hypothesis restricting to twenty-four possibilities the syntactical part of lexical expressions. Combining these possibilities, it should be possible to represent essentially any language.

These expressions use the minimalist grammar variation with weak and strong features, which does not change in a great way the conversion complexity, and each expression contains at most two selectors, a move feature and four features all told.

According to our estimations, the conversion of such a grammar, of a lexicon of n expressions, would be at most of the order of $1.2 \times 10^{11} + O(n)$ operations (according to the time analysis), or $8.5 \times 10^6 + O(n)$ rules (according to the space analysis), doubtless with greatly lesser constants, which fits more with the predictable order of size for a human language analysis. . .

Note that, for a computer-based approach, we just need to convert the grammar *once* to have an efficient parsing afterwards.

In the light of the previously covered equivalencies, an interesting experiment would be to compare the efficiency of a conversion into MCFG, followed by the use of Albro's parser, on the one hand, and followed by a (trivial) conversion into an sPRCG G_{RCG} and the use of the Earley-type parser for RCGs. The results should be of the same kind, the latest being of an $O(n^p)$ parsing for a sentence of length n , with p depending upon the characteristics of the grammar².

Another approach would be to examine the procedures used for the conversion and analysis, and see whether the direct parsing of minimalist grammars could be improved to the point of being as efficient.

²According to [Bou98], this complexity is more precisely, at worst, of $O(k(n^{2d(l+1)}))$, for k the number of rules in G_{RCG} , d its maximal dimension (G_{RCG} being an d-sPRCG when G_{MCFG} is a d-MCFG), and l the maximal length of a rule, regarding its right-hand side predicates.

Chapter 5

To sum up...

The first objective of the present work was to list and present several formalisms for syntactical modeling from natural language processing. Indeed, for a researcher, be her a linguist, a psychologist, a neurologist, a computer scientist or a mathematician, the field of computational linguistics may seem very harsh at first, with regard to the large number of specific concepts.

From this point of view, the first part of this report can enable one to find quickly the main characteristics, including the generative power, of a given natural language processing formalism that one has to study or research.

This overview of formalisms is far from exhaustive, but we have compelled ourselves to cover, for every language class layer, some of its most studied representatives. We have also attempted to realize a state of the art of the equivalences between formalisms, using some reference works in that aspect.

Our subject suggested thereafter to focus our research on minimalist grammars and the equivalent formalisms. It appears that these equivalences, along with conversion procedures and the resulting algorithms have recently been the subject of extensive studies, which are probably not yet widely known within the computational linguistics community.

A part of this report is dedicated to the coverage and explanation, in a would-be pedagogical way, of those particular works, so that they might become easier to fathom.

We also tried to assess the accuracy of objections that can be made in regard of the performance of parsing algorithms that these studies have provided, and, concurrently, to determinate which factors could increase the complexity of the parsing or generation of texts using particular instances of the corresponding – for, even as it appears that languages used in human communications are

really included within the set of languages that can be expressed using some of these formalisms (undoubtedly with some modifications), they seem to be highly specialized versions. . .

In the course of the Master thesis for which this report have been written, we have been led to reflect upon the evolution of Natural Language Processing as a scientific field.

It appears that the problem of the syntactical parsing of the sentence and of the modeling of the skill of generating sentences from grammars is on its way to be solved: whether it is a formalism that is derived from minimalist grammars, range concatenation grammars, or an entirely new one, it looks like one (possibly multiple) unified formalism, able to represent the syntax of the whole of human-mind-graspable languages and implementable using simple and efficient algorithms, is nearing its completion.

There would only remain the long, never-ending work that is to model, using grammars instantiating this formalism, each and every existing linguistic phenomena, vocabulary words and grammar and spelling rules binding them, as well as exceptions to these rules and, in the case of French, exceptions to exceptions. . .

From our viewpoint, it is now of greater importance to try and model the aspects of language beyond sentence syntax.

Indeed, research in natural language processing is not as advanced in these fields, that nonetheless are also part of the speaker's linguistic skill: whether it is be *semantics*, that provides, alongside the syntactical structure of the sentence (that deals with words), its logical structure (that deals with concepts) or *pragmatics*, that places the context of every item regarding supposedly known data, numerous mechanisms remain to be implemented in order to make it possible to model the *sense* of words and structures, be it *denotative* (the dictionary definition – that already might be ambiguous) or *connotative* (representing, between others, the various senses associated by logical or phonetical similarity), and to bind them to get the sense of the sentence, the utterance, or the text.

These works should rely upon a robust syntactical analysis, such as might be provided by minimalist grammars: indeed, the syntactical organisation remains the core of the relationships between the concepts expressed by the words; it would avail one nothing to know every sense of every word of a text and to be unable to combine them. Indeed, some of the most challenging aspects of semantics lies within this notion, the *compositionality* of the lexical senses.

Numerous studies in those fields are in progress, and a lot more remain to be undertaken till a complete computer model of the *understanding* of languages is achieved. . .

Appendix A

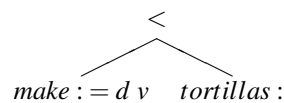
Minimalist Grammars: Tree structures vs. String manipulations

One of the most important aspect of minimalist grammars has intentionally been overlooked in this report, that is their ability to maintain an evolving tree structure, that represents the transformations applied to the analyzed terms. This syntactical tree is based upon both \bar{X} -theory and Chomsky's minimalist program.

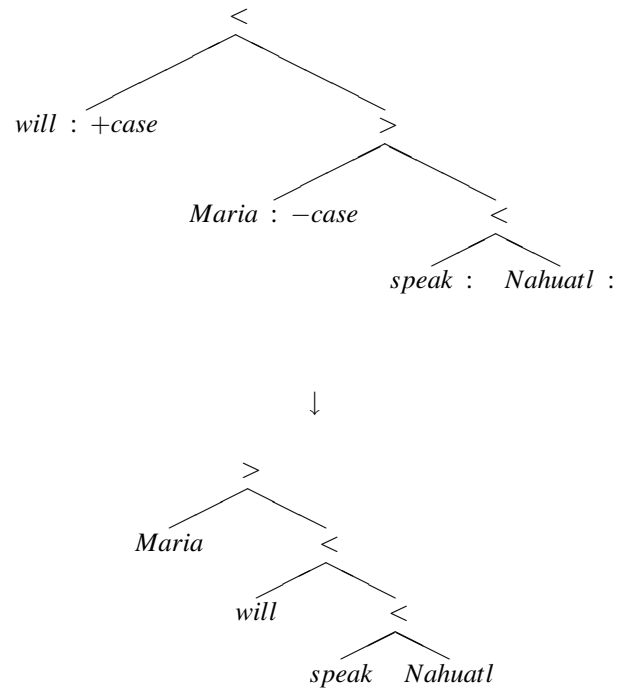
This tree is constructed as derivational rules are added, as the multiple move and merge operations were designed with tree manipulation in mind.

The following examples, inspired from [Sta97b], can allow to easily grasp the concepts involved:

- The merge of entries $make ::= d = d v$ and $tortillas ::= d$ is the tree:



- A move in a derived structure is applied as such:



The final tree structure, after all derivations have been applied, contains, in a condensed form, the syntax of the sentence in a fashion similar to \bar{X} terminology: the binary relation $<$ is used to define a *projection* order, allowing to distinguish between *head*, *specifier* and *complementizer*. It is

easy to see that the tree-structure-enabled minimalist grammars are equivalent to the string version we used, as the operations remain exactly the same, and the manipulation of a syntax tree rather than strings can simply be added, at an asymptotically insignificant cost.

On the other hand, the conversion of minimalist grammars to MCFGs does not preserve that tree structure. MCFGs being entirely constructed over the model of generative grammars, they do not allow for such an easy replacement.

It is possible to construct a syntax tree similar to that of a given minimalist grammar, when parsing using a converted MCFG : as the rules of the MCFG correspond exactly to possible derivations within the original grammar, we need to keep a record of the rules used during the parsing and provide the corresponding "derivation tree".

However, it is troublesome to convert a tree-oriented formalism to a string-capable one, and then reconstruct the associated tree.

An interesting study would be to formalize the operations used with the tree structure associated to minimalist grammars as logical MSO statements that can generate Regular Tree Grammars (as Morawietz suggests in [Mor03]). One could then work from there, in order to ultimately produce an efficient formalism that operates directly on syntax trees.

Appendix B

A Glossary of the formalism acronyms used

CFGs: For *Context-Free Grammars*. Classical formalism, among the generative grammars, generating type-2 languages. See Chomsky's Hierarchy, section 2.1.1, p. 3.

GCFGs: For *Generalized Context-Free Grammars*. Extension of CFGs, using tuples of strings, each non-terminal symbol being produced by the application of some function of other non-terminal symbols. Definition: section 3.2.1, p. 15. Detailed in [SMFK91].

HGs: For *Head Grammars*. Extension of CFGs, using string pairs. They are not defined in detail in the present report. Their generative power is equivalent to that of TAGs – see fig. 2.2, p. 7.

hMGs: Or complete minimalist grammars, the original formalisation of minimalist grammars, as introduced in [Sta97b]. Variations of MGs including additional properties, such as head movement. Rules of derivation are added, features can change, but, for the variations grouped under this name, the language class is the same. Pieces of definition: section 3.4.3, p. 25. Weakly equivalent formalisms: MGs, MCFGs, LCFRSs, MCTAGs, sPRCGs.

LCFRSs: For *Linear Context-Free Rewriting Systems*. Variation of GCFGs restricted the scope of the functions used by rules, allowing neither the copy nor the erasing of variables. Definition and examples: section 3.2.4, p. 19. Equivalent formalisms: MGs, hMGs, MCTAGs, MCFGs, sPRCGs. The (strong) LCFRS-sPRCG equivalence is detailed in section 4.1.2, p. 27, and in [Bou98], and the LCFRS-MCFG equivalence is detailed in section 4.1.3, p. 28.

- MCFGs:** For *Multiple Context-Free Grammars*. Variation of GCFGs restraining the scope of the functions used by rules, allowing the erasing, but not the copy, of variables. Definition and examples: section 3.2.3, p. 18. Detailed in [SMFK91]. Weakly equivalent formalisms: MGs, hMGs, LCFRSs, sPRCGs, MCTAGs. The MG-MCFG equivalence is detailed section 4.2, p. 29.
- MCTAGs:** For *Multiple Component Tree Adjoining Grammars*. Variation of TAGs allowing for multiple adjunction possibilities for a given auxiliary tree. Definition: section 3.1.1, p. 13. Weakly equivalent formalisms (for the set-local version): MGs, hMGs, LCFRSs, MCFGs, sPRCGs.
- MGs:** For *Minimalist Grammars*. Generative grammars using expression movement and combination, born from Noam Chomsky’s minimalist program. Definition and examples: section 3.4, p. 21. Weakly equivalent formalisms: hMGs, MCTAGs, LCFRSs, MCFGs (the MG-MCFG equivalence is covered in section 4.2, p. 29).
- PMCFGs:** For *Parallel Multiple Context-Free Grammars*. Variation of GCFGs restraining their scope, allowing for variable copy or erasing. Definition and examples: section 3.2.2, p. 16. Detailed in [SMFK91]. Equivalent formalism: none known, but extended to an intersection closure, they have the same language class as RCGs – see section 4.1.1, p. 27.
- RCGs:** For *Range Concatenation Grammars*. Adapted from constraint-based grammatical systems used e.g., in PROLOG, they generate exactly all polynomial-time parsable languages. Definition and examples: section 3.3, p. 21. Introduced in [Bou98]. Weakly equivalent formalisms: intersection-PMCFGs, see section 4.1.1, p. 27.
- sPRCGs:** For *simple Positive Range Concatenation Grammars*. Variation of RCGs using only positive, non-erasing, non-copying, with no terminal symbols in the right-hand side of rules predicates. Introduced in [Bou98]. Definition and examples: section 3.3.1, p. 21. Weakly equivalent formalisms: MGs, hMGs, MCTAGs, LCFRSs (the LCFRS-sPRCG equivalence is strong, and detailed in section 4.1.2, p. 27).
- TAGs:** For *Tree Adjoining Grammars*. Provides an intuitive representation of the tree structure of the sentence. Definition and examples: section 3.1, p. 11. Introduced in [JLT75]. Weakly equivalent formalisms: HGs.

Appendix C

Internet and bibliographical references

The original version of this work can be found on my research web page, <http://labri.fr/~mery>.

The minimalist grammar conversion algorithms (MGs as well as hMGs) from [Gui04], along with MCFG parsing algorithms, were recently implemented by Peter Lunglöf as a Prolog library, available at <http://www.cs.chalmers.se/~peb/software.html>.

Several utilities, including the parsers mentioned, are listed on Edward Stabler's web page, <http://www.linguistics.ucla.edu/people/stabler/epssw.htm>.

That page also lists numerous other reference works, as well as the state of current research.

Jens Michaelis' works on formal properties of minimalist grammars and their variations are available at <http://tcl.sfs.uni-tuebingen.de/~michael/papers.html>.

Maxime Amblard's papers, on a syntax-semantic interface for minimalist grammars, are available at <http://www.labri.fr/perso/amblard/>.

The sentence logical structure parser conceived and proposed by Richard Moot, that can be used as a syntactic basis for studies in semantics or pragmatics, *Grail*, is available at <http://www.labri.fr/perso/moot/grail.html>.

Bibliography

- [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.
- [Amb05] Maxime Amblard. Counting dependencies and Minimalist Grammars. Technical report, LaBRI, April 2005.
- [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.
- [CS70] John Cocke and Jacob T. Schwartz. Programming languages and their compilers: preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.
- [Ear70] J. Earley. An efficient context-free parsing algorithm. In *Communications of the Association for Computing Machinery*, 1970.
- [Gro97] Annius Victor Groenink. *Surface without Structure: Word order and tractability issues in natural language processing*. PhD thesis, Utrecht University, Utrecht, Netherlands, November 1997.
- [Gui04] Matthieu Guillaumin. Conversions between Mildly Context Sensitive Grammars. 2004.
- [Har01a] Hendrik Harkema. Parsing Minimalist Languages. 2001.
- [Har01b] Henk Harkema. A Characterization of Minimalistic Languages. In *LACL 2001*, June 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.
- [JS97] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In *Handbook of formal languages, vol. 3: beyond words*, pages 69–123. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
- [Kal05] Laura Kallmeyer. A descriptive characterization of multicomponent tree adjoining grammars. In *TALN 2005*, Dourdan, France, 2005.
- [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.
- [Mor03] Frank Morawietz. *Two-Step Approaches to Natural Language Formalisms*. Studies in Generative Grammar. Mouton de Gruyter, 2003.
- [SCD00] William Schuler, David Chiang, and Mark Dras. Multi-component tag and notions of formal power. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 448–455, Morristown, NJ, USA, 2000. Association for Computational Linguistics.
- [SCK⁺03] Edward P. Stabler, Travis C. Collier, Gregory M. Kobele, Yoosook 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.
- [Shi85] Stuart M. Shieber. Evidence against the context-freeness of natural language. In *Linguistics and Philosophy* 8, pages 333–343. D. Reidel Publishing Company, 1985.
- [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.
- [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.
- [Wei88] David Jeremy Weir. *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis, 1988. Supervisor-Aravind K. Joshi.

- [You67] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10:189–208, 1967.

Appendix D

Acknowledgments

A surprisingly large number have contributed, in a great or small but always appreciated way, to my research, studies and writing sessions.

First and foremost, this work couldn't have taken place at all without the everlasting support of my family and friends, among them expatriates Samuel Méril and Frank Morféa, unlucky victim of the University job openings reductions Michel Fournier, engineer-school exiled Nada Ayad, and my fellow students in other fields such as quantum physics or biochemistry who have made me aware of the relative easiness of my task. I also must thank the team and coffee of *La Soucoupe*, the campus cafeteria that has been, given the non-existence of offices allocated to Master students, quickly became my dedicated workplace. The LaBRI, though, has been kind enough to provide me with accommodation during the time of the translation into English of this report.

In addition to providing me with a research subject and their guidance, my referring teachers, Maxime Amblard, Irène Durand and Christian Retoré have given me, during our weekly meetings and long conversations, the most precious moments of this four-month term. Without their questioning, counseling and suggestions, this work would have been critically shortened. It is due to this constant interaction and collaborative work that this report has made it to the light of the day, even if it does not respond to every expectation of the subject as it started out – subject, we now know, that was very ambitious for a Master thesis. It is my utmost pleasure to associate my name to theirs for this research.

Many other people have contributed to this report, beginning with every author of the quoted articles, and in particular Professors Kasami, Seki, Fuji and Matsumura, for their especially complete and instructive article [SMFK91], and Professors Stabler and Michaelis, for their work and quick answers.

I have also come to rely upon software from D. E. Knuth, Jeffrey Mark Sisking, the Free Software Foundation and Apple Computers, among others, and upon the works of Clamp, Yôko Kanno and numerous artists and musicians (mostly from Japan) for moral support.

I would like to give my thanks to everyone I have named or unfairly forgotten, doubtless in the hundreds...

Bruno Mery, November 2006



Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399