On the Complexity of Free Word Orders

Jérôme Kirman, Sylvain Salvati

On the complexity of free word orders

Jérôme Kirman and Sylvain Salvati

LaBRI, CNRS/Université Bordeaux, INRIA, France

Abstract. We propose some extensions of mildly context-sensitive formalisms whose aim is to model free word orders in natural languages. We give a detailed analysis of the complexity of the formalisms we propose.

1 Introduction

Many natural languages present some free word order phenomena. In some sentences, certain words or phrases can be exchanged freely without changing their meanings in an essential way. It is rather usual to model free word order phenomena by means of dependency grammars [6,14]. We here take another approach that is pertaining to the tradition of generative grammars. We try to see how to enrich formalisms that are considered to be *mildly context sensitive* [8] so as to enable them to model free word order phenomena. Even though the capabilities of mildly context sensitive formalisms to model free word orders remains largely unknown [20,12], some evidence based on syntactic structures indicate that they are not well-suited for modeling free word orders in natural languages [2]. Our motivations are thus twofold: first we wish to close the gap between the dependency approaches and the generative approaches to natural language, following a research direction proposed by Kuhlmann [13]; second we wish to see how to increase the expressiveness of well-known mildly context sensitive formalisms so that they can model free word orders. A question related to the second motivation consists in understanding how robust mild context sensitivity is with respect to such extensions. In this paper, we focus on the computational difficulty of the extensions we propose and show that for certain of them the membership problem remains polynomial.

As is well-known and emphasized by formalisms like Abstract Categorial Grammars (ACGs) [5], mildly context sensitive formalisms have a natural counterpart in terms of tree languages, such as regular tree languages, non-duplicating context free tree languages, multiple regular tree languages, tree languages generated by hyperedge replacement grammars... Our approach consists in defining an algebra with letters as basic nullary operators and with two binary operators: (i) an associative and commutative operator that models the possibility of displacing phrases, (ii) an associative operator that models the usual concatenation. Then we use the *tree counterpart* of mildly context sensitive formalisms so as to generate terms over that algebra. Each of these terms, modulo the equational theory of the algebra denotes a finite set of strings. Then the tree languages we construct denote string languages that are the union of the finite languages

denoted by the terms the grammar generates. This amounts to representing certain sentences up to the ordering of certain of their elements while the possible orderings still have the same syntactic tree; this models the fact that the semantics is preserved and is, in our opinion, crucial in modeling free word order phenomena. A side effect, is that the formalisms we define are naturally more expressive than their "string" counterparts.

The idea is rather similar to the one proposed by Muskens [15]; we nevertheless try to be more conservative with respect to formal language theory. In particular, we try to stay as close as possible to mildly context-sensitive formalisms. It is also related to the notion of ID/LP grammar [22], but the grammatical formalisms we propose are more expressive than context-free grammars.

The paper thus defines some extensions of well-known mildly context-sensitive formalisms using the techniques that are related to ACGs. We nevertheless adopt a presentation that is close to that of Multiple Context-Free Grammars (MCFG) so as to emphasize the relation with mildly context-sensitive formalisms. We then give an analysis of the complexity of the formalisms we obtain. Most of them have an NP-complete membership problem, but we manage to define two tractable subclasses, one being in NLOGSPACE and being an extension of regular languages, the other being LOGCFL-complete and being an extension of context-free languages. Concerning the universal membership problem, the complexity ranges from NP-hardness up to EXPTIME-completeness for the formalisms we consider.

2 Words modulo commutation

2.1 An algebra for representing words modulo commutation

We write $[n]$ for the set $\{1, \ldots, n\}$ and given a finite set Σ, we write Σ^* for the set of strings over Σ, ε denoting the empty string.

The set of types *type* is the smallest set containing 0 and so that when α and β are in *type*, $(\alpha \to \beta)$ is also in *type*. As it is usual we consider that the operator $\to$ associates to the right and that $\alpha_1 \to \cdots \to \alpha_n \to 0$ denotes the type $(\alpha_1 \to (\cdots \to (\alpha_n \to 0)\ldots))$. The order $\mathrm{order}(\alpha)$ of a type α is defined by $\mathrm{order}(0) = 1$, $\mathrm{order}(\beta \to \gamma) = \max\{\mathrm{order}(\beta) + 1, \mathrm{order}(\gamma)\}$. The types that have order 2, or *second order types*, are all of the form $0 \to \cdots \to 0 \to 0$. In general, we shall write $0^k \to 0$ for the type defined as $0^0 \to 0 = 0$, $0^{k+1} \to 0 = 0 \to (0^k \to 0)$. Notice that when $k > 0$, $0^k \to 0$ denotes a second order type.

A *signature* Σ is a finite set of typed constants. These constants c^α come with their types written as superscripts. The order of a signature Σ is $\mathrm{order}(\Sigma) = \max\{\mathrm{order}(\alpha) \mid c^\alpha \in \Sigma\}$. In this paper we are going to work only with second order signatures. We assume that for each type α, we have an infinite countable set of λ-variables x^α, y^α, z^α, ... On a signature Σ, we can construct typed λ-terms, $(\Lambda^\alpha(\Sigma))_{\alpha \in type}$ as the smallest sets such that if c^α is in Σ, c^α is in $\Lambda^\alpha(\Sigma)$, if x^α is a variable of type α, x^α is in $\Lambda^\alpha(\Sigma)$, if M is in $\Lambda^{\beta\to\alpha}(\Sigma)$ and N is in $\Lambda^\beta(\Sigma)$, (MN) is in $\Lambda^\alpha(\Sigma)$, and if M is in $\Lambda^\alpha(\Sigma)$, $\lambda x^\beta.M$ is in $\Lambda^{\beta\to\alpha}(\Sigma)$. We

let the order of a term be the order of its type and we are going to work mostly with terms of order at most 2. Notice that variables and constants come with their types, nevertheless, we shall often omit type annotations; we shall also drop parenthesis following the usual conventions of λ-calculus. We write $FV(M)$ for the set of free variables of M. Given a term M, we write $|M|$ for the size of M defined as: $|c| = |x| = 1$, $|MN| = |M| + |N|$, $|\lambda x.M| = |M|$. For a constant a, we write $|M|_a$ for the number of occurrences of the constant a in M; for a set of constants $\mathcal{C}$, we write $|M|_{\mathcal{C}} = \sum_{a \in \mathcal{C}} |M|_a$. A term is linear when for each of its subterms, if it is of the form MN, then $FV(M) \cap FV(N) = \emptyset$ and if it is of the form $\lambda x.M$ then $x \in FV(M)$. Notice that in a linear terms each variable has at most one free occurrence.

We assume the usual notion of λ-calculus (β, η or $\beta\eta$-contraction/reduction/conversion, β-normal form...) and work up to α-conversion. The simultaneous capture avoiding substitution of M_1, ..., M_n respectively for x_1, ..., x_n in M is written $M[M_1/x_1, \ldots, M_n/x_n]$. Given a function σ that maps variables to terms, we write $M.\sigma$ for $M[\sigma(x_1)/x_1, \ldots, \sigma(x_n)/x_n]$ when $FV(M) = \{x_1, \ldots, x_n\}$ in M.

Given a first order signature (all constants in Σ have type 0) Σ, we define $\mathrm{com}(\Sigma)$ to be the signature $\Sigma \cup \{\varepsilon^0, \bullet^{0^2 \to 0}, \otimes^{0^2 \to 0}\}$ where ε^0 is a constant that is not in Σ. Given a closed term M in β-normal form and of type 0 built on $\mathrm{com}(\Sigma)$, we can map it to a string of Σ^* by simply taking the yield of M as follows:

1. $\mathrm{yield}(a) = a$,
2. $\mathrm{yield}(\varepsilon) = \varepsilon$,
3. $\mathrm{yield}(\bullet M_1\, M_2) = \mathrm{yield}(\otimes M_1\, M_2) = \mathrm{yield}(M_1)\mathrm{yield}(M_2)$.

We write $\overline{\Sigma}$ for $\Sigma \cup \{\varepsilon^0\}$. In particular, for a term M, $|M|_{\overline{\Sigma}}$ denotes the number of occurrences of the elements of $\Sigma \cup \{\varepsilon\}$ that occur in M.

Moreover, we define the least congruence over λ-terms built on $\mathrm{com}(\Sigma)$, written $\equiv_c$, that includes $\beta\eta$-conversion and so that:

$$\begin{aligned} \bullet(\bullet M_1\, M_2)\, M_3 &\equiv_c \bullet M_1\, (\bullet M_2\, M_3) && \text{(Assoc. } \bullet) \\ \otimes(\otimes M_1\, M_2)\, M_3 &\equiv_c \otimes M_1\, (\otimes M_2\, M_3) && \text{(Assoc. } \otimes) \\ \otimes M_1\, M_2 &\equiv_c \otimes M_2 M_1 && \text{(Com. } \otimes) \end{aligned}$$

In a nutshell $\bullet$ is associative while $\otimes$ is associative and commutative. We shall take a *flatten notation* for terms in β-normal form of type 0 whose free variables also have type 0. Given a linear λ-term M in normal form and of type 0 built only with $\otimes$ (*resp.* $\bullet$) and the free variables of type 0 x_1, ..., x_n appearing in that order from left to right we write $\{N_1, \ldots, N_n\}$ (*resp.* $N_1 \ldots N_n$) for $M[N_1/x_1, \ldots, N_n/x_n]$. We shall also, given a term N, write N^k to denote $\underbrace{N \ldots N}_{k \text{ times}}$.

Now given a closed term M of type 0, it denotes a finite language $\mathcal{L}(M) = \{\mathrm{yield}(N) \mid M \equiv_c N\}$.

There are other algebras that can be considered so as to represent finite sets of strings. In particular, a rather natural choice consists in using partial

orders generated by series/parallel operations. This amounts to taking $\bullet$ and $\|$ as binary operators to represent the sets. Closed terms of type 0 in normal form are interpreted as languages using a homomorphism as follows:

1. $[\![a]\!] = \{a\}$,
2. $[\![\varepsilon]\!] = \{\varepsilon\}$,
3. $[\![\bullet M_1\, M_2]\!] = \{tu \mid t \in [\![M_1]\!] \wedge u \in [\![M_2]\!]\}$,
4. $[\![\| M_1\, M_2]\!] = \{t_1u_1 \dots t_nu_n \mid t_1 \dots t_n \in [\![M_1]\!] \wedge u_1 \dots u_n \in [\![M_2]\!]\}$.

Such a choice gives some rather different notion of free word order languages:

Lemma 1. *The class of finite languages that can be described with $\bullet$ and $\otimes$ is incomparable with the one that can be described with $\bullet$ and $\|$.*

Proof. The language $\mathcal{L}(\otimes(\bullet\varepsilon\,(\otimes a\, b))\, c) = \{abc, bac, cab, cba\}$ cannot be described with $\bullet$ and $\|$.

The language $[\![\|(\bullet a\, b)\, c]\!] = \{cab, acb, abc\}$ cannot be described with $\bullet$ and $\otimes$. □

We will explain later on the reason why we prefer to take $\bullet$ and $\otimes$ instead of $\bullet$ and $\|$. For the moment we show that the problem of checking whether a word belongs to the finite language described by a term built on $\bullet$ and $\otimes$ is NP-complete.

Lemma 2. *Given a closed term M of type 0 in β-normal form and a word w, deciding whether w is in $\mathcal{L}(M)$ is NP-complete.*

Proof. We here only prove the NP-hardness; the proof that the problem is in NP relies on the fact that proving that two closed terms of type 0 in β-normal form are equivalent modulo $\equiv_c$ can be done in PTIME.

We are now going to see that the problem is NP-hard using a reduction of the 3-PART problem. An instance $\mathcal{P}$ of the 3-PART problem is given by a sequence $S = s_1, \dots, s_{3m}$ of natural numbers and a number k so that for $1 \leq i \leq 3m$, $\frac{k}{4} < s_i < \frac{k}{2}$. Such an instance has a solution when there is a partition $S_1, \dots, S_m$ of S so that for every $1 \leq i \leq m$, $\sum_{s \in S_i} s = k$.

So given $\mathcal{P}$ an instance of the 3-PART problem, we construct M and w so that $w \in \mathcal{L}(M)$ iff $\mathcal{P}$ has a solution. The term M is constructed on $\mathrm{com}(\{a, \#\})$. To define M we need first to inductively define on k the terms A_k and H_k by: $A_0 = H_0 = \varepsilon$, $A_{k+1} = \bullet a\, A_k$ and $H_{k+1} = \otimes \#\, H_k$. We then let M and w be defined by (using the flatten notation):

$$M = \{A_{s_1}, \dots, A_{s_{3m}}, H_m\}$$
$$w = (a^k\#)^m.$$

It is rather obvious that w is in $\mathcal{L}(M)$ iff there is M' such that $M' \equiv_c M$, M' is of the form:

$$\{A_{s_{\sigma(1)}}, A_{s_{\sigma(2)}}, A_{s_{\sigma(3)}}, \#, \dots, A_{s_{\sigma(3m)}}\#, \varepsilon\}$$

and $w = \mathrm{yield}(M') = a^{k_0}\# \dots a^{k_{m-1}}\#$ where $k_i = k = s_{\sigma(3i+1)} + s_{\sigma(3i+2)} + s_{\sigma(4i)}$ (notice that the fact that $M' \equiv_c M$ implies that σ is a permutation of $[1, 3m]$). From this it easily follows that w is in $\mathcal{L}(M)$ iff $\mathcal{P}$ has a solution. □

2.2 Semilinearity

Given a finite set Σ, the set of vectors of dimension Σ is $\mathbb{N}^\Sigma$. Given v in $\mathbb{N}^\Sigma$, and a in Σ, we write $v.a$ for the value that v associates to a; we write $v_1 + v_2$ for the sum of two vectors; for a in Σ, we also denote 1_a the vector so that for b in Σ, $1_a.b = 1$ when $b = a$ and $1_a.b = 0$ otherwise. We also write $\|v\|$ for $\max\{v.a \mid a \in \Sigma\}$. We now introduce the notion of *linear* and *semilinear* sets.

Definition 1. *Given a finite set Σ, and v_0, v_1,..., v_n in $\mathbb{N}^\Sigma$, $\mathrm{lin}(v_0, \ldots, v_n)$ denotes the set:*

$$\mathrm{lin}(v_0, \ldots, v_n) = \{v_0 + \sum_{k=1}^{n} k_i v_i \mid k_1, \ldots, k_n \in \mathbb{N}\}$$

A subset V of $\mathbb{N}^\Sigma$ is said linear over Σ *either when it is empty or when there exists v_0, v_1,..., v_n in $\mathbb{N}^\Sigma$ so that $V = \mathrm{lin}(v_0, \ldots, v_n)$.*

A subset V of $\mathbb{N}^\Sigma$ is said semilinear over Σ *when it is a finite union of linear sets over Σ.*

We are now going to see that for a fixed linear set $V = \mathrm{lin}(v_0, \ldots, v_n)$, given k in $\mathbb{N}$ we can compute in $\mathrm{NSPACE}(\log(\|v\|))$ whether a vector v is in V.

Lemma 3. *Given a finite set Σ, we fix a linear set $V = \mathrm{lin}(v_0, \ldots, v_n)$, the problem:*

INPUT *v in $\mathbb{N}^\Sigma$*
OUTPUT *yes when v is in V and no when v is not in V*

can be solved in $\mathrm{NSPACE}(\log(\|v\|))$.

Proof. When a number p is smaller than $\|v\|$, it can be represented in space $\log(\|v\|)$. Thus a vector v' in $\mathbb{N}^\Sigma$ so that $\|v'\| \leq \|v\|$ can be represented in space $|\Sigma| \log(\|v\|)$. Now v is in V iff $v = v_0$ or there is i in $[1, n]$ so that $v - v_i$ is in V. When $\|v'\| \leq \|v\|$, we also have $\|v' - v_i\| \leq \|v\|$ so that the new vector can also be represented in space $|\Sigma| \log(\|v\|)$. This characterization thus yields a non-deterministic algorithm that executes in space $\mathcal{O}(\log(\|v\|))$. □

Corollary 1. *Given a finite set Σ, we fix a semilinear set V, the problem:*

INPUT *v in $\mathbb{N}^\Sigma$*
OUTPUT *yes when v is in V and no when v is not in V*

can be solved in $\mathrm{NSPACE}(\log(\|v\|))$.

Proof. Since V is semilinear, it is a finite union of linear sets. The algorithm consists in choosing non-deterministically one of the linear sets composing V and then check whether v is in this linear set. From lemma 3 this can be done in $\mathrm{NSPACE}(\log(\|v\|))$. □

Definition 2. *Given a word w in Σ^*, we write $\psi(w)$ for the* Parikh image of *w, i.e. the vector v of $\mathbb{N}^\Sigma$ such that $v.a$ is the number of a occurring in w.*

Given a language L included in Σ^*, we say that L is semilinear *when the set $\psi(L) = \{\psi(w) \mid w \in L\}$ is semilinear.*

3 Commutative λ-grammars

We are now going to work with the usual mildly context sensitive grammatical formalisms as term generating devices that will produce sets of closed terms of type 0 over the signature $\mathrm{com}(\Sigma)$. Thus a grammar G is going to have a term language $\mathcal{T}(G)$ and a string language $\mathcal{L}(G)$ so that $\mathcal{L}(G) = \bigcup_{t\in\mathcal{T}(G)} \mathcal{L}(t)$. In the next sections, we are going to study the computational complexity of the universal membership and the membership problems for those grammars.

The presentation we are going to give of a grammar is going to be rather general and in line with the usual definition of MCFGs. We thus define the grammars as bottom-up generative devices. We will use some typed predicates A, B, C, ... as non-terminals. We shall write their types as a list of types $[\alpha_1, \ldots, \alpha_n]$. Given a first order signature Σ, we are going to build derivations by means of *Σ-inference rules* of the form:

$$A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

where:

1. the $x_{i,j}$ are pairwise distinct λ-variables so that if B_i has type $[\beta_{i,1}, \ldots, \beta_{i,n_i}]$, then $x_{i,j}$ has type $\beta_{i,j}$,
2. if A has type $[\alpha_1, \ldots, \alpha_n]$ then M_1, ..., M_n are linear λ-terms in normal form built on $\mathrm{com}(\Sigma)$ and respectively of type α_1, ..., α_n
3. the variables $x_{i,j}$ have at most one occurrence in the M_k's (the rule is *non-deleting* when each variable has exactly one occurrences in the M_k's),
4. the free variables of the M_k's are the variables $x_{i,j}$.

Definition 3. *A* commutative λ-grammar *G is a tuple $(\mathcal{N}, \Sigma, R, S)$ so that:*

1. *$\mathcal{N}$ is a finite set of type* non-terminals*,*
2. *Σ is a first order signature of* terminals*,*
3. *R is a finite set of* Σ-inference rules*,*
4. *S is a non-terminal of $\mathcal{N}$ with type* $[0]$*.*

A grammar is non-erasing *when the rules in R are all non-erasing.*

A derivation judgment *is of the form $\Gamma \vdash_G A(M_1, \ldots, M_n)$ where*

$$\Gamma = B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

where the $x_{i,j}$ are pairwise distinct variables whose types are determined by the B_i. The derivation judgments are derived as follows; given a rule of R:

$$A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

if for every $1 \leq i \leq p$, $\Gamma_i \vdash_G B_i(P_{i,1}, \ldots, P_{i,n_i})$ is derivable[1]*, then $\Gamma_1, \ldots, \Gamma_p \vdash_G A(N_1, \ldots, N_n)$ is derivable where N_k is obtained by substituting $P_{i,j}$ for $x_{i,j}$ in M_k and then normalizing the obtained term. The grammar G defines a term language $\mathcal{T}(G) = \{M \mid\vdash_G S(M)\}$ and a string language $\mathcal{L}(G) = \bigcup_{M\in\mathcal{T}(G)} \mathcal{L}(M)$.*

[1] We assume that the variables in the Γ_i are pairwise distinct. If they were not, a renaming would resolve the problem.

We are not going to study all possible commutative λ-grammars that definition 3 allows to define. We are going to use restrictions on the possible types that non-terminals may have and also on the shapes the rules may have. Nevertheless, those restrictions are harmless in terms of expressive power. Due to results in [11], for every term language $\mathcal{T}$ so that there is a λ-grammar G for which $\mathcal{T}(G) = \mathcal{T}$, there will be a λ-grammar G' satisfying the weakest restriction that we will consider and so that $\mathcal{T}(G')$ is also equal to $\mathcal{T}$. The reason why we take those restrictions into account is to make a precise study of the complexities of the membership and the universal membership problems for the grammars satisfying those restrictions. Moreover those restrictions shall allow us to make the connection with various classes of grammars that are used to capture mildly context sensitive languages.

The restrictions are as follows:

CREG every non-terminal has type $[0]$ and the rules are of the form $A[\mathrm{op}\, a\, x] \leftarrow B[x]$ for $\mathrm{op} \in \{\bullet, \otimes\}$,
CCFG every non-terminal has type $[0]$,
CMG every non-terminal has type $[0^k \to 0]$ for some k,
CMREG every non-terminal has type $[0, \ldots, 0]$,
CMCFG every non-terminal has type $[0^{k_1} \to 0, \ldots, 0^{k_p} \to 0]$, for some $k_1, \ldots, k_p$.

When we forbid the use of $\otimes$ for these restrictions we obtain as string languages for **CREG** exactly the class of regular languages, for **CCFG**, the class of context free languages, for **CMG**, the class of languages definable with non-duplicating macro grammars, for **CMREG** and **CMCFG** we obtain languages definable by multiple context-free grammars. If we are concerned with the term languages these classes of grammars define, we obtain for **CREG**, right-branching regular tree languages, for **CCFG**, regular tree languages, for **CMG**, non-duplicating context free tree languages, for **CMREG**, multiple regular tree languages, and for **CMCFG**, tree languages that are definable by hyperedge replacement grammars.

The reason why we do prefer to use the operators $\bullet$ and $\otimes$ rather than the pair $\bullet$ and $\|$ is that the class corresponding to **CREG** we would obtain in that context would coincide with the class of shuffle languages [21]. A problem is that the class obtained by intersecting shuffle languages with regular languages contains languages which are not semilinear [7]. More importantly the rational cone generated by shuffle languages (*i.e.* the class of languages obtained by rational transduction of shuffle languages) is the set of recursively enumerable languages [1]. These results seem to indicate that the class of languages based on the operator $\|$ are not suitable for modeling natural languages. We claim here that on the contrary the rational transductions of languages that are definable by commutative λ-grammars are not only recursive languages, and are also semilinear. We leave the proof of this claim for some future publication. We can nevertheless notice that the languages that those formalisms define are all semilinear.

Theorem 1. *For every commutative λ-grammar G, the language $\mathcal{L}(G)$ is semilinear.*

In the following sections we are going to be interested in two different problems related to the classes of grammars we have just defined.

Definition 4. *The* membership problem *is as follows; we fix a grammar G:*

INPUT *a word w,*
OUTPUT *yes when w is in $\mathcal{L}(G)$ and no otherwise.*

The universal membership problem *is as follows:*

INPUT *a word w, and a grammar G,*
OUTPUT *yes when w is in $\mathcal{L}(G)$ and no otherwise.*

The main difference between the membership and the universal membership problems is that the grammar G is part of the input of the latter while it is not for the former. Thus, constants coming from the grammar are considered as mere constants in the complexity of the membership problem while they are not in the universal membership problem. For instance, the membership problem for the languages defined by a formula of Monadic Second Order Logic is that of the recognition of a finite state machine and is in LOGSPACE while the universal membership is PSPACE-complete.

4 Universal membership

4.1 Universal membership complexity of CMG CMREG CMCFG

We here give an algorithm that solves the recognition problem for **CMCFG** and as **CMCFG** subsume all the classes of grammars we are interested in, we obtain a recognition algorithm for all of them. This algorithm is going to work in EXPTIME in general and is closely related to the one presented in [9] for MCFGs. The idea behind the algorithm consists mainly in trying to find a derivation using directly the rules of the grammar. So given a grammar $G = (\mathcal{N}, \Sigma, R, S)$, and $w \in \Sigma^*$, the algorithm consists in guessing a term M so that $w = \text{yield}(M)$ and M is in $\mathcal{T}(G)$. One of the difficulties is that, due to the possibility of using ε, M may be of arbitrary size. In order to tackle this difficulty we introduce a simple rewrite system over terms built on $\text{com}(\Sigma)$. This rewrite system is based on the two rules

$$\bullet\varepsilon\,\varepsilon \to_\varepsilon \varepsilon \qquad \otimes\,\varepsilon\,\varepsilon \to_\varepsilon \varepsilon$$

This rewriting system is obviously terminating and confluent, moreover its union with β-contraction also yields a terminating and confluent relation. We may associate to a λ-term M a unique ε-normal form and a unique $\beta\varepsilon$-normal form. It is also easy to see that the following Lemma holds:

Lemma 4. *Given a closed term M of type 0 built on $\text{com}(\Sigma)$, we have:*

1. *if $M \to_\varepsilon^* N$ then $\mathcal{L}(M) = \mathcal{L}(N)$,*
2. *if $w = \text{yield}(M)$ and M is in $\beta\varepsilon$-normal form then $|M|_\varepsilon \leq |w|$ and $|M| \leq 4|w| - 1$.*

Proof. The first item of the Lemma is obvious, for the second statement, in case $w = \text{yield}(M)$ and M is in $\beta\varepsilon$-normal form , it can easily be established by induction on M that $|M|_\varepsilon \leq |w|$, then as M can be seen as a binary tree with at most $2|w|$ leaves, it follows that it contains at most $4|w| - 1$ nodes which establishes the second identity. □

Lemma 5. *Given a linear λ-term, if $M \xrightarrow{k}_\varepsilon N$, then $|M| = |N| + 2k$.*

Proof. It suffices to remark that when $M \to_\varepsilon N$ then $|M| = |N| + 2$ and iterate this identity. □

Lemma 6. *Given two second order linear terms M and N in $\beta\varepsilon$-normal form, if all the variables in $FV(M)$ are second order and σ is a substitution so that for every x, $\sigma(x)$ is linear and $M.\sigma =_{\beta\varepsilon} N$, then:*

1. $|M|_\Sigma + \sum_{x \in FV(M)} |\sigma(x)|_\Sigma = |N|_\Sigma$
2. *for every x, $|\sigma(x)|_\varepsilon \leq |N|_\varepsilon$.*

Proof. The first item of the Lemma is a simple consequence of the linearity of the terms. The second comes from the fact that when contracting a β-redex one may create at most one ε-redex, then analyzing reductions of second order redices we obtain the inequalities. □

Lemma 7. *Given N a term in $\beta\varepsilon$-normal form of type $0^k \to 0$, if $|N|_{\overline{\Sigma}} = l$ then $|N| = 2(k + l) - 1$.*

Proof. Here, $N = \lambda x_1 \dots x_k.P$ and P can be seen as a binary tree with $k + l$ leaves; k of its leaves being the variables $x_1, \dots, x_k$ and l other leaves coming from $\overline{\Sigma}$. □

Given a commutative λ-grammar $G = (\mathcal{N}, \Sigma, R, S)$, and a word w, the algorithm consists in:

1. guessing a term M in $\beta\varepsilon$-normal form so that $w = \text{yield}(M)$,
2. check whether there is N in $\mathcal{T}(G)$ so that $N \xrightarrow{*}_\varepsilon M$.

Using Lemma 4, it is easy to see that the first step can be achieved in NP.

For describing how the algorithm solves the second step we introduce fresh typed constants $\perp_\alpha$, and items of the form $\langle A, P_1, \dots, P_n \rangle$ where A is a non-terminal of $\mathcal{N}$ of type $[\alpha_1, \dots, \alpha_n]$ and for $1 \leq i \leq n$, either P_i is a linear λ-term in $\beta\varepsilon$-normal form of type α_i or $P_i = \perp_{\alpha_i}$. An item $\langle A, P_1, \dots, P_n \rangle$ is *G-valid* iff there is a tuple of terms $N_1, \dots, N_n$ so that $\vdash_G A(N_1, \dots, N_n)$ is derivable, and for $1 \leq i \leq n$, if $P_i \neq \perp_{\alpha_i}$, then P_i is the ε-normal form of N_i. The algorithm relies on a procedure that decides whether an item is G-valid; it calls it to check whether $\langle S, M \rangle$ is G-valid. Notice that there is M in $\beta\varepsilon$-normal so that $w = \text{yield}(M)$ and $\langle S, M \rangle$ is G-valid iff w is in $\mathcal{L}(G)$.

We are going to present the procedure that establishes whether an item $\langle A, P_1, \dots, P_n \rangle$ is G-valid by means of an alternating Turing machine that works

in PSPACE. This means that this algorithm can be implemented using a Turing machine that computes in EXPTIME [3]. The machine stores on its working tape the item $\langle A, P_1, \ldots, P_n \rangle$ it is trying to prove G-valid, then it chooses a rule in R:

$$A(M_1 \ldots M_n) \leftarrow B_1(x_{1,1} \ldots x_{1,n_1}) \ldots B_p(x_{p,1} \ldots x_{p,n_p})$$

Let, for $1 \leq k \leq n$, $X_k = \{x_{i,j} \mid 1 \leq j \leq n_i \wedge x_{i,j} \in FV(M_k)\}$, $X = \bigcup_{k \in [n] \wedge P_k \neq \bot} X_k$ and let $Y = \{x_{i,j} \mid 1 \leq i \leq p \wedge 1 \leq j \leq n_i\} - X$. The algorithm guesses a substitution σ such that:

1. for k in $[n]$, if $P_k \neq \bot$, $M_i.\sigma \xrightarrow{*}_{\beta\varepsilon} P_i$,
2. $\sigma(x_{i,j})$ is in $\beta\varepsilon$-normal form when $x_{i,j}$ is in X,
3. $\sigma(x_{i,j}) = \bot$ when $x_{i,j}$ is in Y.

From Lemmas 6 and 7, for every $x_{i,j}$ of type $0^{k_{i,j}} \to 0$, $|\sigma(x_{i,j})| \leq 2(k_{i,j} + |P_i|_{\overline{\Sigma}}) - 1$ from which we can conclude that when the algorithm tries to check whether $\langle S, M \rangle$ is G-valid, the item it stores on its working tape is always of the form $\langle A, P_1, \ldots, P_n \rangle$ with $\sum_{i=1}^{n} |P_i| \leq n|M| + \sum_{i=1}^{n} k_i \leq 4n|w| + \sum_{i=1}^{n} k_i$. This implies that the item can be stored in PSPACE and that moreover, the substitution σ can be found in PSPACE; the algorithm thus makes each transition in PSPACE. Obviously the algorithm is correct and complete, and therefore the membership problem can be solved in EXPTIME [3]. As MCFGs are as special kind of **CMCFG**, and as the universal membership problem for MCFGs is EXPTIME-hard [9], we obtain that the universal membership problem for **CMCFG** is EXPTIME-complete.

Theorem 2. *The universal membership problem for **CMCFG*** EXPTIME*-complete.*

It we restrict our attention to non-deleting **CMREG** we can show that this algorithm runs in alternating PTIME in a similar way as the case of LCFRSs is treated in in [9]. As non-deleting **CMREG** subsume non-deleting LCFRSs whose universal membership is PSPACE-complete, we obtain that:

Theorem 3. *The universal membership problem for non-deleting **CMREG*** PSPACE*-complete.*

For grammars in **CMG** we know from [9] that the universal membership problem is PSPACE-hard, but we do not know whether it is PSPACE-complete or EXPTIME-complete even for the non-deleting case.

4.2 Universal memberhip complexity for CREG and CCFG

We prove here that the universal membership problems for **CREG** and **CCFG** are NP-complete.

Lemma 8. *The universal membership problem for **CREG** is* NP*-hard.*

Proof. We use a reduction of the NP-complete Exact 3-cover problem, or X3C. An instance $\mathcal{P}$ of the X3C problem is a finite set $\mathcal{M} = \{a_1, \ldots, a_{3m}\}$ and a set $\mathcal{F} = \{S_1, \ldots, S_n\}$ so that for every i in $[1, n]$, $S_i \subseteq \mathcal{M}$ and $S_i = \{a_{i,1}, a_{i,2}, a_{i,3}\}$. The problem $\mathcal{P}$ has a solution iff there is a subset $\mathcal{F}'$ of disjoint elements of $\mathcal{F}$ so that their union is $\mathcal{M}$. Given an instance of X3C such as $\mathcal{P}$, we let $G_{\mathcal{P}} = (\{S\}, \mathcal{M}, R, S)$ be the **CREG** grammar so that the rules of R are precisely the rules of the form $S(\otimes a_{i,1}(\otimes a_{i,2}(\otimes a_{i,3}\, x))) \leftarrow S(x)$ for i in $[1, n]$ or the rule $S(\varepsilon) \leftarrow$. Now it is easy to see that $a_1 \ldots a_{3m}$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. □

As an immediate corollary we obtain:

Corollary 2. *The universal membership problem for **CCFG** is* NP*-hard.*

Theorem 4. *The universal membership problem for **CREG** and **CCFG** is* NP*-complete.*

Proof. To prove this we only need to find an algorithm in NP that solves this problem. Let us suppose that we are given a **CCFG** $G = (\mathcal{N}, \Sigma, R, S)$, and a word w. Then given a word w, we guess a term M in $\beta\varepsilon$-normal form so that $\text{yield}(M) = w$ and there is M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$. Lemma 4 shows that $|M| \leq 4|w| - 1$, thus checking that $\text{yield}(M) = w$ can be done in polynomial time. It thus remains to prove that checking the existence of M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$ can also be done in polynomial time. For this, we remark that, for A in N, deciding whether there is a term P so that $P \xrightarrow{*}_{\varepsilon} \varepsilon$, and $\vdash_G A(P)$ is derivable can be decided in polynomial time. For each A in $\mathcal{N}$ for which there is P so that $P \xrightarrow{*}_{\varepsilon} \varepsilon$ and $\vdash_G A(P)$ we add a rule $A(\varepsilon) \leftarrow$ to R. The new grammar $G' = (\mathcal{N}, \Sigma, R', S)$ recognizes M iff there is M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$. Then checking whether M is in $\mathcal{T}(G')$ can obviously be done in polynomial time showing that the universal membership problem for **CCFG** is in NP. □

5 Membership problem: the polynomial cases

Before we turn to proving that the membership problems for the grammars **CREG** and **CCFG** are tractable, we first introduce some technical notions that will be useful in both cases.

First of all we assume that the rules of the **CCFG**'s over the set of terminals Σ we consider are of one of the forms:

$$A(\text{op}\, x\, y) \leftarrow B(x),\, C(y) \qquad \text{where op} \in \{\bullet, \otimes\} \qquad (1)$$

$$A(\text{op}\, x\, a) \leftarrow B(x) \qquad \text{where op} \in \{\bullet, \otimes\} \qquad (2)$$

$$A(a) \leftarrow \qquad \text{where } a \in \Sigma \cup \{\varepsilon\} \qquad (3)$$

Usual constructs allow to transform any **CCFG** G into a **CCFG** G' respecting this restriction and so that $\mathcal{T}(G) = \mathcal{T}(G')$ (and thus $\mathcal{L}(G) = \mathcal{L}(G')$). We can also make similar transformations that allow us to work without loss of generalization with **CCFG** whose rules are only of the forms (1) and (3).

Definition 5. *Given a* ***CCFG*** $G = (\mathcal{N}, \Sigma, R, S)$*, we define* $\mathbf{c}(G) = (\mathcal{N}, \Sigma, R', S)$ *so that* R' *is the set of rules of* R *that are like the rules (1) or (2) with* $\mathrm{op} = \otimes$*. We then define the language* $\psi(G, A)$ *to be the subset of* $\mathbb{N}^{\mathcal{N}\cup\Sigma}$ *so that*

$$\{\psi(\mathrm{yield}(M.\sigma)) \mid B_1(x_1), \ldots, B_n(x_n) \vdash_{G'} A(M) \wedge \sigma(x_i) = B_i\}$$

The set of vectors in $\psi(G, A)$ represent the commutative strings over $(\mathcal{N} \cup \Sigma)^*$ that can be constructed with G from the non-terminal A. It can intuitively be understood as the set of *commutative sentential forms* that are derivable from A.

A simple consequence of Theorem 1 is that the sets $\psi(G, A)$ are semilinear. Moreover, it is easy to get an actual representation those sets using Parikh's construction [16].

Lemma 9. *Given a* ***CCFG*** $G = (\mathcal{N}, \Sigma, R, S)$*, the sets* $\psi(G, A)$ *are semilinear.*

5.1 Membership problem for CREG

In this section, we are going to see that the membership problem for **CREG** is in NLOGSPACE.

$$\frac{}{\langle S, 0, S, 0\rangle}\ \textsc{init} \qquad \frac{\langle A, 0, A, i\rangle \quad A(\bullet\, a_{i+1}\ B) \in R}{\langle B, 0, B, i+1\rangle}\ \textsc{scan}$$

$$\frac{\langle A, v, B, i\rangle \quad v + 1_B \in \psi(G, A) \quad v = \psi(w, i, j)}{\langle B, 0, B, j\rangle}\ \textsc{commutative scan}$$

$$\frac{\langle A, v, B, i\rangle \quad B(\otimes\, a\ x) \leftarrow C(x) \in R \quad \|v + 1_a\| \leq n}{\langle A, v + 1_a, C, i\rangle}\ \textsc{commutative guess}$$

$$\frac{\langle A, v, B, j\rangle \quad B(a) \leftarrow\ \ \in R \quad v + 1_a = \psi(w, j, n)}{\top}\ \textsc{success}$$

Fig. 1. The NLOGSPACE recognition algorithm for **CREG**

Theorem 5. *The membership problem for* ***CREG*** *is in* NLOGSPACE*.*

Proof. Let us fix a **CREG** grammar $G = (\mathcal{N}, \Sigma, R, S)$, given a word $w = a_1 \ldots a_n$ we solve the problem whether $w \in \mathcal{L}(G)$ by using items either of the form $\top$, to denote that w has been recognized, or of the form $\langle A, v, B, i\rangle$ where A, B are in $\mathcal{N}$, v is in $\mathbb{N}^{\Sigma}$ and $\|v\| \leq |w|$, and $0 \leq i \leq |w|$. For describing the rules of the algorithm we are going to use the notation with $\psi(w, i, j)$ to denote the Parikh image of the string $a_{i+1} \ldots a_j$ when $i < j$ and the vector 0 otherwise. The algorithm is described by the inference rules of Figure 1. In the rule COMMUTATIVE SCAN we implicitly assume that in the sum $v + 1_B$ the vector v is injected in $\mathbb{N}^{\Sigma\cup\mathcal{N}}$ by giving it value 0 on the coordinates in $\mathcal{N}$ and 1_B is in $\mathbb{N}^{\Sigma\cup\mathcal{N}}$.

We are first going to see that deciding whether an item is derivable can be done in NSPACE($\log(|w|)$). First of all we know the items can all be represented in space $\mathcal{O}((|\Sigma|+1)\log(|w|))$; thus, to prove that the algorithm can be run in NSPACE($\log(|w|)$), it suffices that each rule can be executed within NSPACE($\log(|w|)$). The rules INIT and SCAN pose no problem. The rules COMMUTATIVE GUESS and SUCCESS require checking whether a certain vector has a norm smaller than $|w|$ which can be easily done in NSPACE($\log(|w|)$). Finally the rule COMMUTATIVE SCAN requires to checking the equality of two vectors whose norm is smaller than $|w|$, which can be done in NSPACE($\log(|w|)$), and also that one of those vectors is in the semilinear set $\psi(G, A)$ which according to corollary 1 can be done in NSPACE($\log(|w|)$). This finally shows that the algorithm described by the inference rules of figure 1 can be executed in NLOGSPACE.

It is then easy to see that $\langle A, v, B, i\rangle$ is derivable iff $A(x) \vdash_G S(M)$ so that $a_1 \dots a_i x$ is in $\mathcal{L}(M)$, and $B(x) \vdash_G A(N)$ so that $N = \otimes b_1(\cdots(\otimes b_r x)\cdots)$ with $b_1, \dots, b_r$ in $\Sigma \cup \{\varepsilon\}$ and $\psi(\text{yield}(N)) = v$. From this it easily follows that $\top$ is derivable iff w is in $\mathcal{L}(G)$. □

5.2 Membership problem for CCFG

In this section we are going to see that the membership problem for **CCFG** is LOGCFL-complete. As every language definable by a context-free grammar can be seen as a **CCFG** we know that the membership problem for **CCFG** is LOGCFL-hard. We are thus going to see that this problem is actually in LOGCFL.

Theorem 6. *The membership problem for **CCFG** is* LOGCFL*-complete.*

Proof. We consider a **CCFG** $G = (\mathcal{N}, \Sigma, R, S)$ whose rules are of the forms (1) and (3). We are going to show that there is an alternating Turing machine (ATMs) that recognizes the word w in $\mathcal{L}(G)$ in a space $\mathcal{O}(\log(|w|))$ whose accepting computation trees have size $\mathcal{O}(|w|^{|\mathcal{N}|+1})$ (the size of a computation tree is the number of nodes that are visited by the machine together with all the possible successors of existential nodes). The results of Ruzzo [18], imply then that the problem is in LOGCFL. Before we describe the algorithm for a given word $w = a_1 \dots a_n$, for $0 \leq i \leq j \leq n$ we write $w[i, j]$ for $a_{i+1} \dots a_j$ when $i < j$ and ε when $i = j$. We describe the machine by means of the inference system of figure 2 that works with items of the form $\langle v, i, j\rangle$ where v is in $\mathbb{N}^{\mathcal{N}}$ so that $\|v\| \leq |w|$ and $0 \leq i \leq j \leq |w|$. The machine accepts a word $w = a_1 \dots a_n$ if it can derive the item $\langle 1_S, 0, n\rangle$. The working tape of the machine contains the item that is the current goal of the inference system. The choice of a rule is made by an existential choice, and when a rule is chosen, all the premises are needed to be proven using alternation. For a given item we need to give an upper bound on the number of instances of the rules that can be applied to pursue the derivation that is polynomial in $|w|$. The key observation to prove this consists in noticing that there are $\mathcal{O}(\log(|w|)^{|\mathcal{N}|+2})$ possible items so that each rule may have only have polynomially many instances while the machine is trying to recognize w. Moreover each item can be represented in space $\mathcal{O}((|\mathcal{N}|+2)\log(|w|)) = \mathcal{O}(\log(|w|))$.

Finally it is easy to see that a tree that derives an item $\langle v, i, j\rangle$ with the inference system of figure 2 contains exactly $2(j-i-1)-1$ nodes, which implies that the accepting derivations of $\langle S, 0, n\rangle$ have size $2|w|-1$. An accepting computation tree of the machine is thus made of a derivation tree of $\langle S, 0, n\rangle$ together with all the possible rules that can be applied at a given node of that tree; therefore such a tree has size $\mathcal{O}(\log(|w|)^{|N|+2}|w|)$. This finally shows that the machine is implementing a LOGCFL algorithm. It is rather straightforward to prove that

$$\frac{\langle v_1, i, j\rangle \quad \langle v_2, j, k\rangle \quad 0 < v_1 \quad 0 < v_2 \quad \|v_1 + v_2\| \leq |w|}{\langle v_1 + v_2, i, k\rangle}\ \text{COMMUTATIVE COMBINE}$$

$$\frac{\langle v, i, j\rangle \quad v \in \psi(G, A)}{\langle 1_A, i, j\rangle}\ \text{COMMUTATIVE REDUCTION}$$

$$\frac{\langle 1_B, i, j\rangle \quad \langle 1_C, j, k\rangle \quad A(\bullet x\, y) \leftarrow B(x),\, C(y)}{\langle 1_A, i, k\rangle}\ \text{COMBINE}$$

$$\frac{A(a) \quad a = w[i, j]}{\langle 1_A, i-1, i\rangle}\ \text{CONSTANT}$$

Fig. 2. The LOGCFL recognition algorithm for **CCFG**

w is in $\mathcal{L}(G)$ iff $\langle S, 0, n\rangle$ is derivable with the inference system of figure 2. For this it suffices to remark that $\langle v, i, j\rangle$ iff there is $u = A_1 \dots A_k$ in $\mathcal{N}^*$ so that $\psi(u) = v$, and $w[i, j] = u_1 \dots u_k$ with, for all $1 \leq i \leq k$, $\vdash_G A_i(u_i)$. This finally shows that the membership problem is in LOGCFL. □

6 Membership problem: the intractable cases

In this section we are going to see that the membership problems for **CMG**, **CMREG**, **CMCFG** are NP-hard. As every commutative λ-grammar that is a **CMG** or a **CMREG** is also a **CMCFG**, we only need to prove that the membership problems for **CMG** and **CMREG** are NP-hard.

For this we shall use reductions from the 3-PART problem that we have already used to prove Lemma 2.

Lemma 10. *The membership problem for **CMREG** is* NP*-hard.*

Proof. We are going to construct a grammar **CMREG** $G = (\mathcal{N}, \Sigma, R, S)$ so that for every instance $\mathcal{P}$ of 3-PART there is a word $w_{\mathcal{P}}$ of size linear within the size of $\mathcal{P}$ so that $w_{\mathcal{P}}$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. We construct G as follows: $\mathcal{N} = \{A, S\}$ with A of type $[0, 0, 0, 0]$ and S of type $[0]$; $\Sigma = \{a, b, \#\}$; then the rules in R (using the flatten notation) are: $S(\{x_1, x_2, x_3, x_4, y\}) \leftarrow A(x_1, x_2, x_3, x_4)\, S(y)$, $S(\varepsilon) \leftarrow$, $A(a\, x_1, x_2, x_3, b\, x_4) \leftarrow A(x_1, x_2, x_3, x_4)$, $A(x_1, a\, x_2, x_3, b\, x_4) \leftarrow A(x_1, x_2, x_3, x_4)$, $A(x_1, x_2, a\, x_3, b\, x_4) \leftarrow A(x_1, x_2, x_3, x_4)$, and $A(\#, \#, \#, \#) \leftarrow$. It is easy to see

that $A(M_1, M_2, M_3, N)$ is derivable iff $M_1 = a^{k_1}\#$, $M_2 = a^{k_2}\#$, $M_3 = a^{k_3}\#$ and $N = b^k\#$ with $k = k_1 + k_2 + k_3$. Then it follows that, $\vdash_G S(M)$ is derivable iff $M = \{a^{k_{1,1}}\#, a^{k_{1,2}}\#, a^{k_{1,3}}\#, b^{k_1}\#, \ldots, a^{k_{n,1}}\#, a^{k_{n,2}}\#, a^{k_{n,3}}\#, b^{k_n}\#, \varepsilon\}$ for some n and for every $1 \leq i \leq n$, $k_i = k_{i,1} + k_{i,2} + k_{i,3}$. Now, given an instance $\mathcal{P}$ of 3-PART that consists in the sequence $S = s_1, \ldots, s_{3m}$ of natural numbers, and a number k so that for $1 \leq i \leq 3m$, $\frac{k}{4} < s_i < \frac{k}{2}$, it is easy to see that $w_{\mathcal{P}} = a^{s_1}\# \ldots a^{s_{3m}}\#(b^k\#)^m$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. □

Lemma 11. *The membership problem for* ***CMG*** *is* NP*-hard.*

Proof. The proof is based on the definition of the same language as in the proof of Lemma 10. □

Lemma 12. *The membership problem for* ***CMCFG*** *is in NP.*

Proof. Let us consider a grammar $G = (\mathcal{N}, \Sigma, R, S)$ and a word w on Σ^*. Using the results of [10] we know that there is a grammar G' that recognizes precisely the set $\{M \mid M \text{ in } \beta\varepsilon\text{-normal form and } \exists N \in \mathcal{T}(G).N \xrightarrow{*}_{\varepsilon} M\}$. Notice that $\mathcal{L}(G') = \mathcal{L}(G)$. We construct an algorithm in NP that checks whether w is in $\mathcal{L}(G')$. It guesses a term M in $\beta\varepsilon$-normal form so that $\text{yield}(M) = w$ and M is in $\mathcal{T}(G)$. Checking that M is in $\mathcal{T}(G)$ can be done in PTIME according to [19]. This completes the proof. □

Theorem 7. *The membership problems for* ***CMG****,* ***CMREG*** *and* ***CMCFG*** *are* NP*-complete.*

7 Conclusion

The classes of grammars we propose in the paper are quite close to the classes of grammars that are mildly context sensitive and should thus offer the same kind of ease in modeling natural languages. Moreover, if we put aside the informal condition *limited cross-serial dependencies*, the languages defined by **CCFG** satisfy the definition of mildly context sensitive languages. The uniform presentation that we have given to our formalisms together with their closeness to usual mildly context sensitive formalisms should allow the definition of some extensions of **CCFG** that are still mildly context sensitive.

Besides closing the picture concerning the universal memberhip problems for commutative λ-grammars, there are other research directions that we shall follow. A first one is to give the characterizations of certain natural families of languages that are generated by the families of languages that are definable with the grammars we proposed, such as the rational cones or the abstract families of languages they generate. A second one is to give a comparison of their expressive power with some known formalisms that share the same properties such as Unordered Vector Grammars [4], or Multiset-Valued LIG [17].

References

1. T. Araki and N. Tokura. Flow languages equal recursively enumerable languages. *Acta Informatica*, 15(3):209–217, 1981.
2. T. Becker, O. Rambow, and M. Niv. The derivational generative power of formal systems or scrambling is beyond LCFRS. Technical Report IRCS-92-38, UPENN, 1992.
3. A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *JACM*, 28(1):114–133, 1981.
4. J. Dassow. Grammars with regulated rewriting. In *Formal Languages and Applications*, pages 249–273. Springer, 2004.
5. P. de Groote. Towards abstract categorial grammars. In ACL, editor, *Proceedings 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155, 2001.
6. Kim Gerdes and Sylvain Kahane. Word order in German: A formal dependency grammar using a topological hierarchy. In *ACL*, pages 220–227, 2001.
7. J. Jedrzejowicz and A. Szepietowski. On the expressive power of the shuffle operator matched with intersection by regular sets. *RAIRO*, 35(04):379–388, 2001.
8. Aravind K. Joshi. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In *Natural Language Parsing*, pages 206–250. CUP, 1985.
9. Y. Kaji, R. Nakanishi, H. Seki, and T. Kasmi. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Trans. Inf. & Syst.*, E 75-D(1):78–88, 1992.
10. M. Kanazawa. Abstract families of abstract categorial languages. In *13th WoLLIC*, ENTCS, pages 65–80, 2006.
11. M. Kanazawa. Second-order abstract categorial grammars as hyperedge replacement grammars. *Journal of Logic, Language and Information*, 19(2):137–161, 2010.
12. M. Kanazawa and S. Salvati. MIX is not a tree-adjoining language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 666–674. ACL, 2012.
13. Marco Kuhlmann and Mathias Möhl. Mildly context-sensitive dependency languages. In *ACL*, 2007.
14. Markéta Lopatková, Martin Plátek, and Vladislav Kubon. Modeling syntax of free word-order languages: Dependency analysis by reduction. In *TSD*, pages 140–147, 2005.
15. R. Muskens. Separating syntax and combinatorics in categorial grammar. *Research on Language and Computation*, 5(3):267–285, 2007.
16. R. Parikh. On context-free languages. *JACM*, 13(4):570–581, 1966.
17. O. Rambow. Multiset-valued linear index grammars: imposing dominance constraints on derivations. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 263–270. ACL, 1994.
18. W.L. Ruzzo. Tree-size bounded alternation. *JCSS*, 21(2):218–235, 1980.
19. S. Salvati. *Problèmes de filtrage et problème d'analyse pour les grammaires catégorielles abstraites*. PhD thesis, INPL, 2005.
20. S. Salvati. Mix is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is solved by a third-order collapsible pushdown automaton. Technical report, INRIA, 2011.
21. A. Shaw. Software descriptions with flow expressions. *Software Engineering, IEEE Transactions on*, 4(3):242–254, 1978.
22. S. Shieber. Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, 2(7):135–154, 1984.