On IO-Copying and Mildly-Context Sensitive Formalisms

Pierre Bourreau[1], Laura Kallmeyer[2], and Sylvain Salvati[1]

[1] Université Bordeaux 1
351, Cours de la Libération
33405 Talence Cedex, France
{bourreau,salvati}@labri.fr
[2] Heine-Heinrich Universität Düsseldorf
Universitätstr. 1
40225 Düsseldorf, Germany
kallmeyer@phil.uni-duesseldorf.de

Abstract. The class of mildly context-sensitive languages is commonly regarded as sufficiently rich to capture most aspects of the syntax of natural languages. Many formalisms are known to generate families of languages which belong to this class. Among them are tree-adjoining grammars, multiple context-free grammars and abstract categorial grammars. All these formalisms have in common that they are based on operations which do not copy already derived material in the course of a derivation. We propose an extension of the class of languages captured by these formalisms that is arguably mildly context-sensitive. This extension is based on a mild use of a copying operation we call IO-substitution.

Keywords: mildly context-sensitive formalisms, abstract categorial grammars, semilinear and constant-growth languages, IO grammars.

1 Introduction

The question of the amount of expressive power a formalism needs so as to adequately model natural language syntax is still open. Nevertheless the notion of *mildly context sensitive languages* that Joshi [Jos85] proposes plays a structuring role in this debate. A class **L** of languages is said to be *mildly context-sensitive* [Jos85, Wei88] if it satisfies the following properties:

1. **L** contains all context-free languages.
2. if L is in **L** then L describes only some restricted crossing dependencies.
3. every language $L \in \mathbf{L}$ satisfies the constant-growth property.
4. for every language $L \in \mathbf{L}$, recognition can be performed in polynomial-time.

Some well-known formalisms are widely regarded as defining mildly context-sensitive classes of languages, like Linear Context-Free Rewriting Systems [VSWJ87] (LCFRSs), Multiple Context-Free Grammars [SMMK91] (MCFGs), Minimalist Grammars [Sta96] (MGs) and set-local multicomponent Tree Adjoining Grammars [Wei88] (MCTAGs). All those formalisms actually define the same class of languages. Nevertheless, we

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 1–16, 2013.

think it is more appropriate to say that a class of languages is mildly context-sensitive rather than speaking of *the* class of mildly context-sensitive languages. Indeed, property 2 is mostly intuitive and not really well formalized. It therefore induces some subjective judgments about whether a class of languages is mildly context-sensitive. Indeed, while MCFLs are usually regarded as mildly context sensitive, this second property is interpreted in [JSW91] as *perhaps [excluding] the so-called language MIX*, which has recently been shown to be in the class of MCFLs [Sal11].

Property 3 defining a mildly context-sensitive class of languages is also open to debate. For example, Weir [Wei88] considers that *the constant-growth property is too weak to capture [the linguistic] intuition fully [...] the slightly stronger property of semilinearity may come closer*. As it happens, all the formalisms we mention above are semilinear. So as to understand what difference it makes to consider languages that satisfy the constant-growth property rather than the semilinearity property, we explore a way of defining interesting families of languages that are not semilinear but that satisfy the constant-growth property. For this goal, we extend the class of LCFRL/MCFL following ideas from [Kal10] by adding to these formalisms a limited capacity of copying during derivations. This approach differs from that of [Kal10] in that it does not try to find the most general way of achieving this goal. In particular, we try to obtain formalisms whose derivations are similar to those of context-free grammars, and as a consequence we secure the decidability of the emptiness problem for the languages we define.

Our proposal is based on an operation that can be traced back to Fischer [Fis68a] and that we call *IO-substitution*. Roughly speaking, an IO-substitution of a language L_2 in a language L_1 amounts, for every word w of L_2, to replacing in every word of L_1 all the occurrences of a given letter x by w. In such an IO-substitution, one can see the words of L_1 as *parametrized by the letter* x, this parameter being instantiated with words taken from L_2. In general, when recursively applied, such an operation gives rise to languages that may have an exponential growth. Interestingly, we show that under certain simple conditions, a class of languages satisfying the constant-growth property is closed under IO-substitution. Thus taking the closure of classes of semilinear languages, such as MCFLs/LCFRLs, under this operation gives rise to classes that have the constant-growth property but which may not be semilinear anymore.

The structure of the paper is as follows: in the first part of the article, we recall the notions of semilinearity and constant-growth for languages. We then introduce the operation of IO-substitution on languages and show how the constant-growth (*resp.* semilinearity) property can be preserved when applied on constant-growth (*resp.* semilinear) languages. Thanks to this result, we define a new class of languages we call $IO(\mathbf{MCFL})$, as a candidate for the biggest mildly context-sensitive class. In the second part of the article, we characterize a grammatical formalism which exactly captures languages in $IO(\mathbf{MCFL})$. Such grammars are particular abstract categorial grammars of second-order, the generated languages of which are tree languages. The given formalism enjoys the property of being analyzable in polynomial-time, which confirms $IO(\mathbf{MCFL})$ as a candidate for being a mildly context-sensitive class of languages.

2 IO-Substitutions, Semilinearity and Constant-Growth

2.1 Basic Definitions

Given a finite set Σ, we write Σ^* for the set of words built on Σ, and ϵ for the empty word. Given w in Σ^*, we write $|w|$ for its length, and $|w|_a$ for the number of occurrences of a letter a of Σ in w. A language on Σ is a subset of Σ^*. Given two languages $L_1, L_2 \subseteq \Sigma^*$, $L_1 \cdot L_2$, the concatenation of L_1 and L_2, is the language $\{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$; the union of L_1 and L_2 is written $L_1 + L_2$.

We write $\mathbb{N}$ for the set of natural numbers. For a given alphabet Σ, $\mathbb{N}^\Sigma$ is the set of vectors whose coordinates are indexed by the letters of Σ. Given $a \in \Sigma$ and $\overrightarrow{v} \in \mathbb{N}^\Sigma$, $\overrightarrow{v}[a]$ denotes the value of the coordinate a of $\overrightarrow{v}$. We write $\overrightarrow{e_a}$ for the unit vector on the a-dimension, *i.e.* the vector such that $\overrightarrow{e_a}[c] = 0$ when $c \neq a$ and $\overrightarrow{e_a}[a] = 1$.

The constant-growth property expresses some constraint about the distribution of the possible length of the words in a language:

Definition 1. *A language $L \subseteq \Sigma^*$ is said to be* constant-growth *if there exist $k, c \in \mathbb{N}$ such that, given $w \in L$ if $|w| > k$, then there is $w' \in L$ such that $|w| < |w'| \leq |w| + c$.*

A class of languages $\mathbf{L}$ *is* constant-growth *when every L in* $\mathbf{L}$ *is constant-growth.*

Most of the mildly context-sensitive classes of languages commonly used verify the stronger property of semilinearity, which is based on the notion of the Parikh image.

Definition 2. *Let us consider a word w in a language $L \subseteq \Sigma^*$. The* Parikh image of w, *written $\overrightarrow{p}(w)$ is the vector of $\mathbb{N}^\Sigma$ such that, for every $a \in \Sigma$, $\overrightarrow{p}(w)[a] = |w|_a$. The Parikh image of L is defined as $\overrightarrow{p}(L) = \{\overrightarrow{p}(w) \mid w \in L\}$.*

Definition 3. *A set V of vectors of $\mathbb{N}^\Sigma$ is called* linear *when there are vectors $\overrightarrow{x_0}, \ldots, \overrightarrow{x_n}$ in $\mathbb{N}^\Sigma$ such that $V = \{\overrightarrow{x_0} + k_1\overrightarrow{x_1} + \ldots + k_n\overrightarrow{x_n} \mid k_1, \ldots, k_n \in \mathbb{N}\}$. V is called* purely linear *when $\overrightarrow{x_0} = \overrightarrow{0}$.*

A set of vectors is called semilinear *(resp.* purely semilinear*) when it is a finite union of linear (resp. purely linear) sets.*

Given two sets of vectors V_1 and V_2, we will denote by $V_1 + V_2$ the set $\{\overrightarrow{v_1} + \overrightarrow{v_2} \mid \overrightarrow{v_1} \in V_1, \overrightarrow{v_2} \in V_2\}$. Similarly, given $k \in \mathbb{N}$ and a set of vectors V, we will write $kV = \{k\overrightarrow{v} \mid \overrightarrow{v} \in V\}$.

Definition 4. *A language L is called* semilinear *when $\overrightarrow{p}(L)$ is a semilinear set. A class of languages* $\mathbf{L}$ *is* semilinear *when all the languages it contains are semilinear.*

Given two alphabets Σ_1 and Σ_2, a string homomorphism h from Σ_1^* to Σ_2^* is a function such that $h(\epsilon) = \epsilon$ and $h(w_1 w_2) = h(w_1)h(w_2)$. Given $L \subseteq \Sigma_1^*$, we write $h(L)$ for the language $\{h(w) \in \Sigma_2^* \mid w \in L\}$. The homomorphism h is *alphabetic* when for each a in Σ_1, $h(a)$ is in Σ_2.

Definition 5. *Let us consider a class* $\mathbf{L}$ *of languages which share the same alphabet Σ. Given an n-ary operation* $\mathsf{op} : (\Sigma^*)^n \mapsto \Sigma^*$, *where $n \in \mathbb{N}$, we say that* $\mathbf{L}$ *is closed under* op *if for every $L_1, \ldots, L_n \in \mathbf{L}$, $\mathsf{op}(L_1, \ldots, L_n) \in \mathbf{L}$.*

In the rest of the paper, we will be particularly interested in **MCFL**, the class of *multiple context-free languages*. This class of languages is known to be mildly context-sensitive and semilinear. It is also closed by homomorphism, intersection with regular sets, union and concatenation [SMMK91]. Note that our study could also be done on other mildly context-sensitive classes of string languages, such as the classes of context free-languages, or yields of tree-adjoining grammars, or well-nested mildly context-sensitive grammars.

2.2 IO-Substitution and Copies

In this section, we introduce an operation on languages, which we will use to define constant-growth languages. This operation, which we call *IO-substitution*, enables a specific kind of copying mechanism, which we want to use so as to capture some languages that do not belong to **MCFL**, but still satisfy the constant-growth property:

Example 1. The language $L_{a,b} = \{(a^n b^n)^m \mid n,m \in \mathbb{N}\}$ is not a **MCFL** (see [Kal10] for a proof). Nevertheless, it is a semilinear language, as its Parikh image is $\{k(1,1) \mid k \in \mathbb{N}\}$, a semilinear set.

The language $L_{count} = \{(a^n d)^m a^n \mid n,m \in \mathbb{N}\}$ is not a semilinear language, but satisfies the constant-growth property: given a word w in L_{count} such that $|w| \geq 0$ it holds that $|w| < a a^{|w|} \leq |w| + 1$.

The language $L_{exp} = \{a^{2^n} \mid n \in \mathbb{N}\}$ is neither semilinear, nor constant-growth.

Interestingly, these three languages are also known to be generated by IO-macro grammars [Fis68b, Fis68a], in which copying operations are allowed in a specific way, which we will not discuss in detail in this article. We here introduce the notion of IO-substitution that allows us to mimic in a non-recursive way the copying operations that are used in IO-macro grammars. This notion of substitution can be compared with the notion of *language substitution* which corresponds to the non-recursive counterpart of the copying operations in OI-macro grammars (another notion of macro grammars defined by Fischer [Fis68b, Fis68a]). While language substitution consists in replacing each occurrence of a letter a in the words of a language L_1 by some word of a language L_2, IO-substitution requires that every such occurrence of a is replaced by the very same word of L_2.

Definition 6. *Let us consider an alphabet Σ, and two languages $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$. Given a word $w \in L_2$, and a symbol $a \in \Sigma$, we define the homomorphism $h_{a,w} : \Sigma^* \mapsto \Sigma^*$ by $h_{a,w}(c) = c$ for $c \in \Sigma - \{a\}$ and $h_{a,w}(a) = w$.*

The language $L_1[a := L_2]_{IO}$ is then defined as $\bigcup_{w \in L_2} h_{a,w}(L_1)$.

Note that $L_1[a := L_2]_{IO} = L_1$ if for every word $w \in L_1$, a has no occurrence in w.

As an example, we can consider the languages $L_1 = a^*$ and $L_2 = ab + c$; the language $L_1[a := L_2]$ is then defined as $(ab + c)^*$.

Definition 7. *Given a class of languages* **L** *built on an alphabet Σ, we define the class $IO_n(\mathbf{L})$ by induction on $n \in \mathbb{N}$ as*

- $IO_0(\mathbf{L}) = \mathbf{L}$ *and*

– $IO_{n+1}(\mathbf{L}) = \bigcup_{L_1,L_2 \in IO_n(\mathbf{L})} \bigcup_{x \in \Sigma} L_1[x := L_2]_{IO}$, *for every* $n \geq 0$

We define $IO(\mathbf{L})$ *as* $\bigcup_{n \in \mathbb{N}} IO_n(\mathbf{L})$.

Notice that $IO(\mathbf{L})$ is the smallest class of languages that contains $\mathbf{L}$ and that is closed under IO-substitution.

Based on this operation, we can build the languages $L_{a,b}$ and L_{count} easily from context-free languages. Indeed, given $L_1 = \{x^n \mid n \in \mathbb{N}\}$ and $L_2 = \{a^m b^m \mid m \in \mathbb{N}\}$, the language $L_{a,b}$ is the language $L_1[x := L_2]_{IO}$. In the same way, $L_{count} = L'_1[x := L'_2]_{IO}$, where $L'_1 = \{(xd)^m x \mid m \in \mathbb{N}\}$ and $L'_2 = \{a^n \mid n \in \mathbb{N}\}$. On the other hand, there is no trivial way of using the IO-substitution operation on multiple context-free languages to generate L_{exp}. One can also remark that L_1, L_2, L'_1 and L'_2 are context-free languages, hence semilinear languages and they satisfy the constant-growth property. It is therefore natural to investigate the conditions under which semilinearity and the constant-growth property are preserved by the IO-substitution operation.

2.3 Preserving the Semilinearity and Constant-Growth Properties

In the previous examples, we have seen that the newly introduced operation of IO-substitution can lead in some particular cases to the construction of semilinear or constant-growth languages when applied to semilinear languages. In this section, we investigate the conditions which lead to the preservation of these properties.

Definition 8. *Given an alphabet* Σ *and* $a \in \Sigma$*:*

– *a set of vectors* $V \subseteq \mathbb{N}^\Sigma$ *is* a-independent *when for every* $\overrightarrow{v} \in V$, $\overrightarrow{v}[a] = 0$.
– *a set of vectors* V *is* a-isolating *if it is of the form* $U + c\{k\overrightarrow{e_a} \mid k \in \mathbb{N}\}$ *where* U *is* a*-independent, and* $c \in \mathbb{N} - \{0\}$.
– *a semilinear set is* a*-isolating when it is a finite union of linear* a*-isolating sets of vectors.*
– *a semilinear language is* a*-isolating when its Parikh image is a semilinear* a*-isolating set of vectors.*

Example 2. Let us consider various languages so as to illustrate a-isolating languages:

– the language $\{x^n \mid n \in \mathbb{N}\}$ is obviously x-isolating, as its Parikh image is $\{n(1) \mid n \in \mathbb{N}\}$.
– the language $abxa^*x^*$ is not x-isolating as its Parikh image is $\{(1,1,1) + n(1,0,0) + m(0,0,1) \mid n,m \in \mathbb{N}\}$ (with $\overrightarrow{e_a} = (1,0,0)$, $\overrightarrow{e_b} = (0,1,0)$ and $\overrightarrow{e_x} = (0,0,1)$).
– the language aba^*x^* is x-isolating, since its Parikh image is $\{(1,1,0) + n(1,0,0) + m(0,0,1) \mid n,m \in \mathbb{N}\} = \{(1,1,0) + n(1,0,0) \mid n \in \mathbb{N}\} + 1\{n(0,0,1) \mid n \in \mathbb{N}\}$.
– the language $(xa)^*x$ is not x-isolating as its Parikh image is $\{(0,1) + n(1,1) \mid n \in \mathbb{N}\}$ (with $\overrightarrow{e_a} = (1,0)$, and $\overrightarrow{e_x} = (0,1)$).

One can remark that, given a word w in a a-isolating language, the number of occurrences of a in w is completely independent from the number of occurrences of any other symbol in w.

We now show that the IO-substitution of a letter a of a semilinear language in a a-isolating language generates a semilinear language.

Lemma 1. *Given $c \in \mathbb{N}$, if V is a purely semilinear set then the set $\{k\overrightarrow{c}\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$ is equal to cV.*

Proof. To prove the Lemma, it suffices to prove that when V is a purely linear set of vectors, then $\{k\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$ is equal to cV. If V is purely linear, let $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ be the vectors such that $V = \{\sum_{i=1}^{n} k_i\overrightarrow{x_i} \mid k_1, \ldots, k_n \in \mathbb{N}\}$. We obviously have $cV \subseteq \{kc\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$, by considering $k = 1$. Moreover given $\overrightarrow{u}$ in $\{kc\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$, we must have $\overrightarrow{u} = kc\overrightarrow{v}$ with $\overrightarrow{v}$ in V, so that $\overrightarrow{u} = kc\sum_{i=1}^{n} k_i\overrightarrow{x_i} = c\sum_{i=1}^{n} kk_i\overrightarrow{x_i}$ and is thus also in cV.

Theorem 1. *Given a semilinear language $L_1 \subseteq \Sigma^*$, which is a-isolating, and a purely semilinear language L_2, the language $L_1[a := L_2]_{IO}$ is semilinear.*

Proof. Since L_1 is semilinear and a-isolating, its Parikh image is a finite union of linear a-isolating sets $V_1, \ldots, V_n$. It is then easy to show that the Parikh image of $L_1[a := L_2]_{IO}$ is the finite union of $V'_1, \ldots, V'_n$ where, for every $1 \leq i \leq n$, $V'_i = U_i + \{kc_i\overrightarrow{u} \mid k \in \mathbb{N}, \overrightarrow{u} \in \overrightarrow{p}(L_2)\}$ (where $c_i \in \mathbb{N}$ and U_i is a-independent). Since $\overrightarrow{p}(L_2)$ is purely semilinear, by Lemma 1, $\{kc_i\overrightarrow{u} \mid k \in \mathbb{N}, \overrightarrow{u} \in \overrightarrow{p}(L_2)\} = c_i\overrightarrow{p}(L_2)$ which is purely semilinear. Thus, V'_i is the sum of two semilinear sets and is semilinear. Therefore $\bigcup_{i=1}^{n} V'_i$ is a semilinear set. As a conclusion, $L_1[a := L_2]_{IO}$ is semilinear.

While the semilinearity property is preserved by IO-substitution under some rather restrictive constraints, we show that for certain classes of languages, some of their properties are also satisfied by their closure under IO-substitution. Indeed, it is also easy to see that $L_1[x := L_2]_{IO}$, where L_1 and L_2 are semilinear languages, is of constant-growth while not necessarily semilinear (as shown with the languages in Example 2). Interestingly, the constant-growth property can be verified by closure under IO-substitution on classes of languages which are closed under homomorphism.

Lemma 2. *Given $L_1, L_2 \subseteq \Sigma^*$, if L_1 is an infinite language that is of constant-growth and $L_1 \subseteq L_2$, then L_2 is of constant-growth.*

Proof. Let k and c be as in Definition 1 for L_1. Since L_1 is infinite there are some words w in L_1 such that $|w| > k$. We let w_0 be a shortest word so that $|w_0| > k$. Without loss of generality we may assume that $|w_0| = k + 1$. Then using the fact that L_1 is constant-growth we construct a sequence $(w_i)_{i\in\mathbb{N}}$ such that $|w_i| < |w_{i+1}| \leq |w_i| + c$. Now, given a word w in L_2, if $|w| > k$ we let n be the integer such that $|w_n| \leq |w| < |w_{n+1}|$. By definition of $(w_i)_{i\in\mathbb{N}}$ we have $|w_{n+1}| \leq |w_n| + c$ so that $|w_{n+1}| \leq |w| + c$. Therefore $|w| < |w_{n+1}| \leq |w| + c$, and L_2 is constant-growth.

Lemma 3. *If a class of languages* **L** *is closed under (alphabetic) homomorphism, then $IO_n(\mathbf{L})$ is closed under (alphabetic) homomorphism, for every $n \in \mathbb{N}$.*

Proof. We proceed by induction on n. For $n = 0$, $IO_n(\mathbf{L})$ is closed under (alphabetic) homomorphism by hypothesis. For $n = k + 1$, let us consider an alphabet Σ, two languages $L_1, L_2 \subseteq \Sigma^*$ in $IO_k(\mathbf{L})$, and a (alphabetic) homomorphism $h : \Sigma^* \mapsto \Sigma^*$. Given a symbol $a \in \Sigma$, we define the (alphabetic) homomorphism $g : \Sigma^* \mapsto \Sigma^*$ such that $g(a) = a$ and $g(b) = h(b)$ when $b \in \Sigma - \{a\}$. It is then easy to check that $h(L_1[a := L_2]_{IO}) = g(L_1)[a := h(L_2)]_{IO}$. But, by the induction hypothesis, both $g(L_1)$ and $h(L_2)$ are in $IO_k(\mathbf{L})$ which proves that $h(L_1[a := L_2]_{IO})$ is in $IO_n(\mathbf{L})$.

We are now in a position to prove the following theorem:

Theorem 2. *If a class of languages* $\mathbf{L}$ *is constant-growth and closed under homomorphism, then* $IO_n(\mathbf{L})$ *is constant-growth, for every* $n \in \mathbb{N}$.

Proof. We consider the alphabet Σ on which the languages of $\mathbf{L}$ are built. We show by induction on n that the class $IO_n(\mathbf{L})$ is constant-growth. From Lemma 3, we have that $IO_n(\mathbf{L})$ is closed under homomorphism.

The case $n = 0$ is immediate. For $n = k+1$, given L_1 and L_2 in $IO_k(\mathbf{L})$, we consider several cases. Let us first suppose that L_1 is infinite; then for w in L_2 and $a \in \Sigma$, we define $h_{a,w}$ to be the homomorphism such that $h_{a,w}(a) = w$ and $h_{a,w}(b) = b$ if $b \in \Sigma - \{a\}$. By definition $L_1[a := L_2]_{IO} = \bigcup_{w \in L_2} h_{a,w}(L_1)$. The cases where $L_2 = \emptyset$ or $L_2 = \{\epsilon\}$ are trivial. So let us suppose that there is $w \neq \epsilon$ in L_2, then $h_{a,w}(L_1)$ is infinite and, by induction hypothesis and because $IO_k(\mathbf{L})$ is closed under homomorphism, $h_{a,w}(L_1)$ is constant-growth. As $h_{a,w}(L_1) \subseteq L_1[a := L_2]_{IO}$, Lemma 2 shows that $L_1[a := L_2]_{IO}$ is constant-growth.

Let us now suppose that L_1 is finite. The cases where, for every w in L_1, $|w|_a = 0$ or where L_2 is finite are trivial. So let us consider w in L_1 such that $|w|_a > 0$ and L_2 infinite. Then $\{w\}[a := L_2]_{IO}$ is infinite and we are going to show that $\{w\}[a := L_2]_{IO}$ is constant growth. Let k and c be integers as in Definition 1 for L_2. We consider $l = k|w|_a + |w| - |w|_a$ (*i.e.* the length of w when weighting an occurrence of a by k, and an occurrence of any other letter of Σ by 1), $d = c|w|_a$, and a word $u \in \{w\}[a := L_2]_{IO}$ such that $|u| > l$. This implies that there is v in L_2 such that $u = h_{a,v}(w)$, and which satifies $|u| = |v||w|_a + |w| - |w|_a$. Moreover, $|u| > l$ implies $|v| > k$; but then, because L_2 is constant-growth, there is v' in L_2 such that $|v| < |v'| \leq |v| + c$. Let u' be $h_{a,v'}(w)$; then $|u'| = |v'||w|_a + |w| - |w|_a$ so that $|u| < |u'| \leq |u| + d$. This finally shows that $\{w\}[a := L_2]_{IO}$ is constant-growth. As $\{w\}[a := L_2]_{IO}$ is infinite and $\{w\}[a := L_2]_{IO} \subseteq L_1[a := L_2]_{IO}$, Lemma 2 implies that $L_1[a := L_2]_{IO}$ is constant-growth.

The string languages generated by multiple-context free-grammars, or second-order ACGs, for example, are known to be closed under homomorphism [SMMK91, Kan06], and satisfy the semilinearity property. As a consequence, the closure of these languages by IO-substitution forms classes of languages which are constant-growth.

As a remark, the following example illustrates the need of the condition of closure by homomorphism for the class of languages we close by IO-substitution:

Example 3. The language $L_1 = \{a^n b^{2^m} \mid n, m \in \mathbb{N}\}$ is a constant-growth language (because a^* is constant-growth), but is not a MCFL. The language $L_1[a := \{\epsilon\}]_{IO} = \{b^{2^m} \mid m \in \mathbb{N}\}$ is not constant-growth.

We know prove other desirable closure properties of classes of languages which are preserved by IO-substitution.

Lemma 4. *Given a class of languages* $\mathbf{L}$ *that is closed under alphabetic homomorphism, the following holds:*

1. *if* $\mathbf{L}$ *is closed under union, then so is* $IO(\mathbf{L})$*,*
2. *if* $\mathbf{L}$ *is closed under concatenation, then so is* $IO(\mathbf{L})$*,*

Proof. Lemma 3 shows that $IO_n(\mathbf{L})$ is closed under alphabetic homomorphism.

By induction on n, we show that if L_1 and L_2 are in $IO_n(\mathbf{L})$ then $L_1 \cup L_2$ (*resp.* $L_1 \cdot L_2$) is in $IO(\mathbf{L})$. The base case is given by the fact that $\mathbf{L}$ is closed under union (*resp.* concatenation). So let us suppose that $n = k+1$, given L_1, L_2, L'_1 and L'_2 in $IO_k(\mathbf{L})$ we show that $L_1[a := L_2]_{IO} \cup L'_1[b := L'_2]_{IO}$ (*resp.* $L_1[a := L_2]_{IO} \cdot L'_1[b := L'_2]_{IO}$) is in $IO_n(\mathbf{L})$. We suppose that $L_1 \subseteq \Sigma_1^*$ and $L'_1 \subseteq {\Sigma'}_1^*$. We let a' and b' be two distinct letters so that both a' and b' are not in $\Sigma_1 \cup \Sigma'_1$ and we let h be the alphabetic homomorphism from Σ_1 to $\Sigma_2 = (\Sigma_1 - \{a\}) \cup \{a'\}$ such that $h(a) = a'$ and $h(c) = c$ when $c \neq a$; similarly we define the alphabetic homomorphism g from Σ'_1 to $\Sigma'_2 = (\Sigma_1 - \{b\}) \cup \{b'\}$ such that $h(b) = b'$ and $h(c) = c$ when $c \neq b$. Then it is easy to check that $h(L_1)[a' := L_2]_{IO} = L_1[a := L_2]_{IO}$ and $g(L'_1)[b' := L'_2]_{IO} = L'_1[b := L'_2]_{IO}$ and moreover, with the induction hypothesis, we get that $L_1[a := L_2]_{IO} \cup L'_1[b := L'_2]_{IO} = ((h(L_1) \cup g(L'_1))[a' := L_2]_{IO})[b' := L'_2]_{IO}$ (*resp.* $L_1[a := L_2]_{IO} \cdot L'_1[b := L'_2]_{IO} = ((h(L_1) \cdot g(L'_1))[a' := L_2]_{IO})[b' := L'_2]_{IO}$) is in $IO_n(\mathbf{L})$.

Lemma 5. *If* $\mathbf{L}$ *is a class of languages that is closed under union, alphabetic homomorphism and intersection with regular sets, then* $IO(\mathbf{L})$ *is closed under intersection with regular sets.*

Proof. The proof of Lemma 3 entails that $IO_n(\mathbf{L})$ is closed under alphabetic homomorphism. We prove the theorem by induction on n. Let $L_1, L_2 \subseteq \Sigma^*$ be in $IO_n(\mathbf{L})$ and R be a regular set whose syntactic monoid is $\mathbb{M} = (\mathcal{M}, \cdot, 1)$ and is recognized by $\mathcal{N} \subseteq \mathcal{M}$ using the monoid homomorphism φ. We are going to show that $L_1[a := L_2]_{IO} \cap R$ is in $IO(\mathbf{L})$. For every m in $\mathcal{M}$, we define R_m to be the regular language recognized by $\{m\}$ using φ and we define the monoid homomorphism φ_m from Σ^* to $\mathbb{M}$ so that $\varphi_m(a) = m$ and for every c in Σ, $c \neq a$ implies $\varphi_m(c) = \varphi(c)$. We then let Q_m be the regular language recognized by $\mathcal{N}$ with the monoid homomorphism φ_m. For each m in $\mathcal{M}$, we let $L_m = (L_1 \cap Q_m)[a := L_2 \cap R_m]_{IO}$, it is then easy to see that $L_1[a := L_2]_{IO} \cap R = \bigcup_{m \in \mathcal{M}} L_m$. But by induction hypothesis, for every $m \in \mathcal{M}$, $L_1 \cap Q_m$ and $L_2 \cap R_m$ are in $IO(\mathbf{L})$ so that L_m is in $IO(\mathbf{L})$. From Lemma 4, since $\mathbf{L}$ is closed under union and under alphabetic homomorphism, $IO(\mathbf{L})$ is closed under union. As R is regular, $\mathcal{M}$ is finite, and $\bigcup_{m \in \mathcal{M}} L_m$ (*e.g.* $L_1[a := L_2]_{IO}$) is in $IO(\mathbf{L})$.

As a consequence of the previous Theorems and of the fact that **MCFL** is closed under union, concatenation, homomorphism and intersection with regular sets [SMMK91], we have the following corollary:

Corollary 1. *The class of languages* $IO(\boldsymbol{MCFL})$ *satisfies the following properties:*

1. *it is constant-growth,*
2. *it is closed under homomorphism, union, concatenation and intersection with regular sets.*

In the following section, we seek a grammatical formalism that captures the class $IO(\mathbf{MCFL})$. Additionally, we show that the membership problem in $IO(\mathbf{MCFL})$ can be solved in polynomial-time. This result shows that $IO(\mathbf{MCFL})$ can arguably be considered as a mildly context-sensitive class of languages bigger than **MCFL**.

3 IO-Multiple Context-Free Languages

We have introduced in the previous section the notions of IO-substitution, and of IO-closure of a class of languages. We have showed that $IO(\mathbf{MCFL})$ is of constant-growth. We are now going to give a grammatical formalism that precisely describes $IO(\mathbf{MCFL})$. This will allow us to see that languages in $IO(\mathbf{MCFL})$ have a polynomial (more precisely **LOGCFL**) membership problem. In order to do so, we are going to encode an IO(MCFL) as a second order Abstract Categorial Grammars (ACG) ([dG01, Mus01]).

3.1 Abstract Categorial Grammars

Given a set of atomic types A, the set of simple types $\mathcal{T}(A)$ on A is the smallest set containing A and such that $(\alpha \to \beta)$ is in $\mathcal{T}(A)$ whenever α and β are in $\mathcal{T}(A)$. We take the usual conventions of writing $\alpha_1 \to \cdots \to \alpha_n \to \beta$ instead of $(\alpha_1 \to (\cdots \to (\alpha_n \to \beta)\cdots))$. The order $\mathtt{ord}(\alpha)$ of a type $\alpha \in \mathcal{T}(A)$ is defined as $\mathtt{ord}(\alpha) = 1$ when α is atomic and $\mathtt{ord}(\alpha) = max(1 + \mathtt{ord}(\alpha_1), \mathtt{ord}(\alpha_2))$ when $\alpha = \alpha_1 \to \alpha_2$.

Definition 9. *A* higher-order signature *(HOS)* $\Sigma = (A, C, \tau)$ *is a tuple made of a finite set of atomic types* A*, a finite set of constants* C *and a function* τ *which associates types in* $\mathcal{T}(A)$ *to constants in* C*.*

The order of a higher-order signature $\Sigma = (A, C, \tau)$ is defined as $max_{c \in C}(\mathtt{ord}(\tau(c)))$. Moreover, we say that a HOS $\Sigma = (A, C, \tau)$ is a *tree signature* (*resp.* a *string signature*) if $\mathtt{ord}(\Sigma) = 2$ (*resp.* $\tau(c)$ is of the form $o \to o$ for every $c \in C$).

Given a HOS $\Sigma = (A, C, \tau)$, we define $(\Lambda^{\alpha}_{\Sigma})_{\alpha \in \mathcal{T}(A)}$ as the family of the smallest sets verifying:

1. a variable x^{α} belongs to $\Lambda^{\alpha}_{\Sigma}$
2. a constant $c \in C$ is in $\Lambda^{\tau(c)}_{\Sigma}$
3. if $M \in \Lambda^{\beta}_{\Sigma}$, then $\lambda x^{\alpha}.M \in \Lambda^{\alpha \to \beta}_{\Sigma}$,
4. if $M_1 \in \Lambda^{\beta \to \alpha}_{\Sigma}$, $M_2 \in \Lambda^{\beta}_{\Sigma}$, then $(M_1 M_2) \in \Lambda^{\alpha}_{\Sigma}$.

The terms are typed *a la Church* so that variables are explicitly carrying their types as exponents, but for the sake of readability, we will often omit this typing notation when it is unnecessary to the understanding.

As usual, we write $M_0 M_1 \dots M_n$ instead of $(\dots(M_0 M_1)\dots M_n)$ and $\lambda x_1 \dots x_n.M$ for $\lambda x_1 \dots . \lambda x_n.M$. We take for granted the notions of β-reduction ($\to^*_{\beta}$) and of the normal form $|M|_{\beta}$ of a term M.

Note that given a tree signature Σ, the construction of a term $M \in \Lambda^{o}_{\Sigma}$ does not make use of the rule 3. Its general form is $xM_1 \dots M_n$ which can be interpreted as a tree $x(t_1, \dots, t_n)$ (t_i being the interpretation of M_i as a tree, for every $i \in \{1, \dots, n\}$). Similarly, if Σ is a string signature, a term $M \in \Lambda^{o \to o}_{\Sigma}$ is of the general form $\lambda x.a_1(\dots(a_n x))$, which can be interpreted as a string $a_1 \dots a_n$.

Example 4. Let us consider a tree HOS $(\{o\}, \{\wedge, \vee, 0, 1\}, \tau)$ where $\tau(\wedge) = \tau(\vee) = o \to o \to o$ and $\tau(0) = \tau(1) = o$. The terms in this signature are boolean formulas made of the connectors $\wedge$ and $\vee$, on the values 0 and 1.

The set $FV(M)$ of free variables of a term M is defined as usual. Moreover, a term M is called closed if it contains no free variable.

Definition 10. *Given a HOS $\Sigma = (A, C, \tau)$, a λ-term M is called* linear *if:*

1. *$M = x$ is a variable or $M = c \in C$, or*
2. *$M = \lambda x.N$, $x \in FV(N)$ and N is linear, or*
3. *$M = M_1 M_2$ if M_1 and M_2 are linear and $FV(M_1) \cap FV(M_2) = \emptyset$.*

The term M is called almost affine *if*

1'. *$M = x$ is a variable or $M = c \in C$, or*
2'. *$M = \lambda x.N$ and N is almost affine, or*
3'. *$M = M_1 M_2$ if M_1 and M_2 are almost affine and if $x^\alpha \in FV(M_1) \cap FV(M_2)$, then $\alpha \in A$.*

Note that given a linear term M and a term M' such that $M \to_\beta^* M'$, M' is also linear.

Given two HOS $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$, we say that Σ_1 and Σ_2 are disjoint if $C_1 \cap C_2 = \emptyset$ and $A_1 \cap A_2 = \emptyset$.

Given a constant c, we also define

- $\Sigma_1 \cup \Sigma_2 = (A_1 \cup A_2, C_1 \cup C_2, \tau(c))$ where Σ_1 and Σ_2 are disjoint and

$$\tau(c) = \begin{cases} \tau_1(c) & \text{if } c \in C_1 \\ \tau_2(c) & \text{if } c \in C_2 \end{cases}$$

- $\Sigma_1 - c = (A_1, C_1 - \{c\}, \tau_1')$, where the domain of τ_1' is $C_1 - \{c\}$, and $\tau_1'(c') = \tau_1(c')$, for every $c' \in C_1 - \{c\}$.

Definition 11. *Given two HOS $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$, a homomorphism $\mathcal{H}$ from Σ_1 to Σ_2 is a function that maps $\mathcal{T}(A_1)$ to $\mathcal{T}(A_2)$, $\Lambda^\alpha_{\Sigma_1}$ to $\Lambda^{\mathcal{H}(\alpha)}_{\Sigma_2}$ for every $\alpha \in \mathcal{T}(A_1)$ and verifies:*

1. *$\mathcal{H}(\alpha \to \beta) = \mathcal{H}(\alpha) \to \mathcal{H}(\beta)$,*
2. *$\mathcal{H}(\lambda x^\alpha.M) = \lambda x^{\mathcal{H}(\alpha)}.\mathcal{H}(M)$, $\mathcal{H}(MN) = \mathcal{H}(M)\mathcal{H}(N)$ and $\mathcal{H}(x^\alpha) = x^{\mathcal{H}(\alpha)}$,*
3. *$\mathcal{H}(c)$ is a closed β-normal λ-term of $\Lambda^{\mathcal{H}(\tau_1(c))}_{\Sigma_2}$.*

Finally, an Abstract Categorial Grammar $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ is a tuple where:

1. $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$ are HOS, respectively called the *abstract* and the *object* signatures of G.
2. $\mathcal{H}$ is a homomorphism from Σ_1 to Σ_2, called *the lexicon*.
3. $s \in A_1$ is *the distinguished type*.

An ACG defines two languages:

- its *abstract language* as $\mathcal{A}(\mathcal{G}) = \{M \in \Lambda^s_{\Sigma_1} \mid FV(M) = \emptyset \text{ and } M \text{ is linear}\}$,
- its *object language* as $\mathcal{O}(\mathcal{G}) = \{M \in \Lambda^{\mathcal{H}(s)}_{\Sigma_2} \mid \exists M' \in \mathcal{A}(\mathcal{G}),\ |\mathcal{H}(M')|_\beta = M\}$.

An ACG is called *linear* (*resp. almost affine*) when for every c in Σ_1, $\mathcal{H}(c)$ is linear (*resp.* almost affine). It is called a *second-order* ACG when Σ_1 is a second-order signature. It is called a *tree-ACG* when Σ_2 is a tree signature and $\mathcal{H}(s)$ is atomic.

It is known that the string languages generated by second-order linear ACGs are precisely **MCFL** ([dGP04] and [Sal06]). In particular the yields of the tree-languages generated by second-order linear ACGs are MCFLs, the yield of a tree being the concatenation of its leaves from left to right. Given a tree language L we write yL for the string language obtained by taking the yields of its trees. In particular given a tree ACG, G, we write $y\mathcal{O}(G)$ for the string language of the yields of the trees in $\mathcal{O}(G)$.

Example 5. Let us consider the tree signatures $\Sigma_1 = (\{s, t_1, t_2\}, \{c, c_1, c_2, c_3, c_4\}, \tau_1)$ and $\Sigma_2(\{o\}, \{a, b, c, d, \epsilon, e, f\}, \tau_2)$ where:

- $\tau_1(c) = t_1 \to t_2 \to s$, $\tau_1(c_1) = t_1 \to t_1$, $\tau_1(c_2) = t_1$, $\tau_2(c_3) = t_2 \to t_2$ and $\tau_1(c_4) = t_2$.
- $\tau_2(a, b, c, d, \epsilon) = o$, $\tau_2(e) = o \to o \to o$ and $\tau_2(f) = o \to o \to o \to o \to o$

We define the second-order tree-ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ such that:

- $\mathcal{H}(t_1) = \mathcal{H}(t_2) = (o \to o \to o) \to o$, $\mathcal{H}(s) = o$
- $\mathcal{H}(c_2) = \mathcal{H}(c_4) = \lambda z.z\epsilon\epsilon$
- $\mathcal{H}(c_1) = \lambda Pz.P(\lambda x_1 x_2.z(eax_1)(ecx_2))$ and $\mathcal{H}(c_3) = \lambda Pz.P(\lambda x_1 x_2.z(ebx_1)(edx_2))$
- $\mathcal{H}(c) = \lambda P_1 P_2.P_1(\lambda x_1 x_3.P_2(\lambda x_2 x_4.f x_1 x_2 x_3 x_4))$

Then, a term is in the abstract language of this ACG if it is of the form

$$c\,\underbrace{(c_1(\ldots(c_1}_{n}\,c_2)\ldots))\,\underbrace{(c_3(\ldots(c_3}_{m}\,c_4)\ldots))$$

for some $n, m \in \mathbb{N}$.

The trees in $\mathcal{O}(G)$ are therefore of the following form:

$$f\,\underbrace{(ea(\ldots(ea\epsilon)\ldots))}_{n}\,\underbrace{(eb(\ldots(eb\epsilon)\ldots))}_{m}\,\underbrace{(ec(\ldots(ec\epsilon)\ldots))}_{n}\,\underbrace{(ed(\ldots(ed\epsilon)\ldots))}_{m}$$

Finally, the yield of this language is the language $\{a^n b^m c^n d^m \mid n, m \in \mathbb{N}\}$, which is known to be a MCFL.

We finish this section by mentioning the following complexity results, which will be used in the next section. We recall that **LOGCFL** is the set of problems which can be reduced in logarithmic space into the problem of recognizability of context-free grammars. This class is known to be a subclass of the problems solvable in polynomial-time (see [Ven87] for more details).

Theorem 3 ([Yos06, Kan07]). *The membership problem of a second-order almost affine ACG is in* ***LOGCFL****.*[1]

3.2 IO-MCFGs as Almost Affine ACGs

As mentioned in the previous section, abstract categorial grammar gives a general framework to speak about **MCFL** as the strings generated by ACGs or, alternatively, as

[1] A polynomial recognizer of second-order almost affine ACGs is given in [BS11]; this algorithm is not known yet to be in **LOGCFL**.

the string languages generated by some tree grammars (which are linear second-order tree-ACGs). In what follows, we seek the characterization of some tree-ACGs which generate tree languages such that their yields form exactly IO-MCFLs. While the construction we give can be applied to string-ACGs, we use tree-ACGs in order to prove the membership problem is **LOGCFL**; indeed the grammars constructed in this way are almost affine tree-ACGs, for which such a complexity result is known [Kan07, Yos06].

Intuitively, given two multiple context-free languages $L_1 \subseteq (\Sigma \cup \{x\})^*$ and $L_2 \subseteq \Sigma^*$, we can first remark that the letter x may have many occurrences in $w_1 \in L_1$. Given a tree-ACG $G_1 = (\Sigma_1, \Sigma_2, \mathcal{H}, s_1)$ such that $y\mathcal{O}(G_1) = L_1$, the idea is to consider x not as a constant in the trees derived by G_1, but as a variable which can be substituted by a word $w_2 \in L_2$ (*i.e.* a tree in a tree-ACG G_2 such that $y\mathcal{O}(G_2) = L_2$). Therefore, we need to build a tree-ACG G_1' such that $t \in \mathcal{O}(G_1)$ iff $\lambda x.t \in \mathcal{O}(G_1')$, and then use a constant $\mathbf{c} : s_1' \to s_2 \to s$ such that $\mathcal{H}(\mathbf{c}) = \lambda x_1 x_2 . x_1 x_2$ to simulate the IO-substitution of x by L_2 into L_1, where s_1' and s_2 are the distinguished types of G_1' and G_2 respectively.

Remark that, because x will now be considered as a variable, the contexts appearing in the left-hand side of a production rule in G_1' will not be linear as in the case of linear ACGs defining MCFLs. But x being a leaf, its type is $\tau(x) = o$, and the contexts will therefore be almost affine, which still ensures the membership problem belongs to **LOGCFL**, hence to the class of problems solvable in polynomial-time.

Definition 12. *Given a second-order ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$, where $\Sigma_i = \{A_i, C_i, \tau_i\}$ for $i \in \{1,2\}$, and $x \in C_2$, we define the second-order ACG $\mathsf{abs}(G, x) = (\Sigma_1, \Sigma_2 - \{x\}, \mathcal{H}', s)$ as follows:*

- *given a type $a \in A_1$, $\mathcal{H}'(a) = \tau_2(x) \to \mathcal{H}(a)$*
- *for every $c \in C_1$, given $\tau_1(c) = \alpha_1 \to \ldots \to \alpha_n \to a$, and $\mathcal{H}(c) = \lambda y_1 \ldots y_n . M$, $\mathcal{H}'(c) = \lambda z_1 \ldots z_n x . |\mathcal{H}(c)(z_1\, x) \ldots (z_n\, x)|_\beta$*

Note that this construction transforms the constant x of the object signature into a variable. Moreover, one should remark that given a constant $c \in C_2$, the free occurrences of x in $\mathcal{H}(\mathbf{c})$ are bound by a λ-abstraction in $\mathcal{H}'(\mathbf{c})$. Finally, we should note that $\mathcal{A}(\mathsf{abs}(G)) = \mathcal{A}(G)$.

Example 6. Let us consider the second-order tree-ACG $G_1 = \{\Sigma_1, \Sigma, \mathcal{H}_1, s_1\}$ where $\Sigma_1 = (\{s_1\}, \{c_1, c_2\}, \tau_1)$ where $\tau_1(c_1) = s_1$, $\tau_1(c_2) = s_1 \to s_1$, and $\Sigma = (\{o\}, \{e, x, \epsilon\}, \tau)$, where $\tau(x) = \tau(\epsilon) = o$, $\tau(e) = o \to o \to o$; moreover $\mathcal{H}_1(s_1) = o$ and $\mathcal{H}_1(c_1) = \epsilon$, $\mathcal{H}_1(c_2) = \lambda y.cxy$.

The abstract language of this grammar is made of trees of the form $\underbrace{c_2(\ldots(c_2}_{n} c_1)\ldots)$, where $n \in \mathbb{N}$.

The tree language derived by G_1 is $\{\underbrace{cx(cx(\ldots(cx}_{n} \epsilon)\ldots)) \mid n \in \mathbb{N}\}$.

The grammar $\mathsf{abs}(G_1, x) = \{\Sigma_1, \Sigma_2 - \{x\}, \mathcal{H}_1', s_1\}$ is such that:

$$\mathcal{H}_1'(c_1) = \lambda x.\epsilon$$
$$\mathcal{H}_1'(c_2) = \lambda zx.|(\lambda y.cxy)(zx)|_\beta = \lambda zx.cx(zx)$$

and derives tree contexts of the form $\lambda x.\{\underbrace{cx(cx(\ldots(cx}_{n} \epsilon)\ldots))$, where $n \in \mathbb{N}$.

The transformation given by $\mathtt{abs}$ intuitively results in considering x as a variable, on which substitutions can be applied. This is made explicit in the following theorem.

Lemma 6. *Given a second-order ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$, the language $O(\mathtt{abs}(G, x))$ is equal to $\{\lambda x.M \in \Lambda_{\mathcal{H}'(s)}(\Sigma_2) \mid M \in O(G)\}$.*

Proof. G being a second-order ACG, so is G'. Moreover, G and G' share the same abstract language. Let us consider a term $M \in O(G)$. By definition, there exists a term N in $\mathcal{A}(G)$, such that $|\mathcal{H}(N)|_\beta = M$. We prove by induction on N that $|\mathcal{H}'(N)|_\beta = \lambda x.M$. The general form of N is $cN_1 \dots N_p$, for some $p \in \mathbb{N}$. If $p = 0$, then $\tau_1(c) = s$, and by construction, we obtain $\mathcal{H}'(s) = \tau_2(x) \to \mathcal{H}(s)$ and $\mathcal{H}'(c) = \lambda x.|\mathcal{H}(c)|_\beta$. Otherwise, by induction hypothesis, $|\mathcal{H}'(N_i)|_\beta = \lambda x.|\mathcal{H}(N_i)|_\beta = P_i$ such that P_i is a term the type of which is $a_i \in A_1$, as the grammar is a second-order ACG. Then $P = \mathcal{H}'(cN_1 \dots N_p) = \mathcal{H}'(c)P_1 \dots P_p$. Given the general form $\lambda y_1 \dots y_n x.\mathcal{H}(c)(y_1 x) \dots (y_n x)$ of $\mathcal{H}'(c)$, we obtain

$$\begin{aligned} P &=_\beta \lambda x.|\mathcal{H}(c)(P_1 x) \dots (P_n x)|_\beta \\ &=_\beta \lambda x.|\mathcal{H}(c)(|\mathcal{H}(c_1)|_\beta \dots |\mathcal{H}(c_n)|_\beta)|_\beta \\ &=_\beta \lambda x.|\mathcal{H}(N)|_\beta \end{aligned}$$

Given two MCFLs L_1 and L_2, we are now in a position to build the ACG which produces the substitution of x by L_2 in L_1. This operation is simulated by application in the lambda-calculus. Indeed, it suffices to consider the tree-ACGs G_1 and G_2 which have languages L_1 and L_2 as respective yields. We then build $\mathtt{abs}(G_1, x)$ and add a constant the image of which is $\lambda x_1 x_2.x_1 x_2$. Because the IO-substitution substitutes every occurrence of a symbol x by a word, we must constrain the corresponding x in the ACG to be of type o.

First, let us consider that two abstract categorial grammars $G_1 = (\Sigma_{11}, \Sigma_{12}, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_{21}, \Sigma_{22}, \mathcal{H}_2, s_2)$ are disjoint if Σ_{11} and Σ_{21} are disjoint.

Definition 13. *A tree-ACG G is called a* tree-IO(ACG) *if*

1. *G is a linear tree-ACG or*
2. *there exist two disjoint tree-IO(ACGs) $G_1 = (\Sigma_1, \Sigma, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_2, \Sigma, \mathcal{H}_2, s_2)$, a constant x in the HOS $\Sigma = (A, C, \tau)$ that satisfies $\tau(x) \in A$, such that, given $\mathtt{abs}(G_1, x) = (\Sigma_1, \Sigma - \{x\}, \mathcal{H}', s_1)$, then $G = (\Sigma', \Sigma, \mathcal{H}, s)$ where:*
   (a) $\Sigma' = \Sigma_1 \cup \Sigma_2 \cup \Sigma_s$ where $\Sigma_s = (\{s_1, s_2, s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and, given $\Sigma_i = (A_i, C_i, \tau_i)$ for $i \in \{1, 2\}$, $c \notin C_1 \cup C_2$, $s \notin A_1 \cup A_2$.
   (b) $\mathcal{H}'(e) = \mathcal{H}_i(e)$ if e belongs to the abstract signature of G_i (where $i \in \{1, 2\}$) and $\mathcal{H}'(c) = \lambda x_1 x_2.x_1 x_2$.

We now prove that this formalism exactly captures tree languages the yields of which are IO-MCFLs:

Theorem 4. *A language L is a IO(**MCFL**) iff there exists a tree IO(ACG) such that $L = yO(G)$.*

Proof. If L is in $IO(\mathbf{MCFL})$, then for some n in $\mathbb{N}$, L is in $IO_n(\mathbf{MCFL})$. We prove the Theorem by induction on n. When $n = 0$, L is a **MCFL**, and from [dGP04], there exists a linear tree-ACG G such that $yO(G) = L$, and G is a tree-IO(ACG) by definition. Conversely, for G a linear tree-ACG, $yO(G)$ is a IO-MCFL ([Kan10], [Sal06]).

Now, suppose there exist two languages L_1, L_2 in $IO_n(\mathbf{L})$, such that $L = L_1[x := L_2]_{IO}$. By induction hypothesis, there exist two tree-IO(ACGs) $G_1 = (\Sigma_1, \Sigma, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_2, \Sigma, \mathcal{H}_2, s_2)$, which without loss of generality can be assumed to be disjoint, such that $yO(G_1) = L_1$ and $yO(G_2) = L_2$. Moreover, x is a constant of $\Sigma = (A, C, \tau)$, $\tau(x) = o$. According to Lemma 6, $O(\mathtt{abs}(G_1, x)) = \{\lambda x.M \in \Lambda_{\Sigma}^{\mathcal{H}'(s)} \mid M \in O(G_1)\}$. We consider the grammar $G = (\Sigma', \Sigma, \mathcal{H}, s)$ where:

1. $\Sigma' = \Sigma_1 \cup \Sigma_2 \cup \Sigma_s$ where $\Sigma_s = (\{s_1, s_2, s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and $c \notin C_{11} \cup C_{21}$, $s \notin A_{11} \cup A_{21}$.
2. $\mathcal{H}'(e) = \mathcal{H}_i(e)$ if e belongs to the abstract signature of G_i (where $i \in \{1, 2\}$) and $\mathcal{H}'(c) = \lambda x_1 x_2 . x_1 x_2$.

A term M is recognized by this grammar iff there exist $M_1 \in O(\mathtt{abs}(G_1, x))$ and $M_2 \in O(G_2)$ such that $M = |M_1 M_2|\beta$ which is the result of substituting every occurrence of x in M_1 by M_2. Finally, we can conclude $yO(G) = \{h_{x,w_2}(w_1) \mid w_1 \in L_1, w_2 \in L_2\}$. Conversely, we prove that the yield of the language of a tree-IO(ACG) is an IO-MCFL, with similar arguments.

Example 7. Let us consider the grammar G_1 given in the previous example, and a grammar $G_2 = \{\Sigma_2, \Sigma', \mathcal{H}_2, s_2\}$, where:

- $\Sigma_2 = (\{s_2\}, \{c_1', c_2'\}, \tau_2)$ and $\tau_2(c_1') = s_2$, $\tau_2(c_2') = s_2 \to s_2$
- $\Sigma' = (\{o\}, \{a, b, d\}, \tau')$ such that $\tau'(a) = \tau'(b) = o$ and $\tau'(d) = o \to o \to o \to o$
- $\mathcal{H}_2(s_2) = o$, and $\mathcal{H}_2(c_1') = \epsilon$, $\mathcal{H}(c_2') = \lambda y.dayb$

The yield of $O(G_2)$ is $\{a^n b^n \mid n \in \mathbb{N}\}$.

By considering the grammar $abs(G_1)$ in Example 6, we build the grammar $G_{a,b} = \{\Sigma_1 \cup \Sigma_2 \cup \Sigma_s, \Sigma \cup \Sigma' - \{x\}, \mathcal{F}, s)$ where $\Sigma_s = (\{s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and $\mathcal{F}(e) = \mathcal{H}_1'(e)$ (*resp.* $\mathcal{F}(e) = \mathcal{H}_2(e)$) if e is a constant in Σ_1 (*resp.* in Σ_2), and $\mathcal{F}(c) = \lambda x_1 x_2 . x_1 x_2$. This grammar is a tree-IO(ACG). Moreover, the yield of the tree language generated by this grammar is $L_{a,b} = \{(a^n b^n)^m \mid n, m \in \mathbb{N}\}$.

Corollary 2. *The membership problem of an IO(**MCFL**) is* **LOGCFL**.

Proof. This is direct consequence of Theorem 3, as we build almost affine ACGs which capture $IO(\mathbf{MCFL})$.

4 Conclusion

In this paper, we presented the operation of IO-substitution on languages, which introduces some form of copying mechanism. We investigated how the properties of semilinearity and constant-growth can be preserved by this operation, and exhibited a new family of languages, which we call $IO(\mathbf{MCFL})$. This class is not semilinear, but is constant-growth. Moreover, it has a membership problem that is polynomial and it

contains context-free languages. It can be thus considered as mildly context-sensitive. Using abstract categorial grammars, we show how to define actual grammars that capture $IO(\mathbf{MCFL})$. In the meantime we also proved that IO-substitution preserves certain combinations of the closure properties of classes of languages: closure under homomorphism, union, concatenation and intersection with a regular set.

Nevertheless, the class of $IO(\mathbf{MCFL})$ was not proved to be closed under inverse homomorphism, and we conjecture that it is not. Clearly the class of $IO(\mathbf{MCFL})$ is strictly weaker than the class of languages derived by CNL-LMGs in [Kal10], in particular because CNL-LMGs contain languages that are intersections of context-free languages. While $IO(\mathbf{MCFL})$ and CNL-LMGs share some properties, $IO(\mathbf{MCFL})$ and CNL-LMG use different implementations of copying: in $IO(\mathbf{MCFL})$, copying is based on copying certain objects that have already been derived; in CNL-LMGs copying is the result of checking the equality of substrings that are derived independently. In $IO(\mathbf{MCFL})$, the reason why we do not go beyond constant-growth languages is due to a relative independence between the recursive nature of derivations and copying. It is likely that we may find a less syntactic characterization of second-order constant-growth ACG by studying more carefully how copying and recursion may interact. Yet another question consists in giving a characterization of the Parikh images of $IO(\mathbf{MCFL})$ that still enjoy some properties of linearity. Finally, some linguistic examples should be given in order to give arguments of whether the IO-substitution operation is needed to give account of the syntactic structure of languages or not. One could, for instance, use IO-substitution to simulate deletion of material as in gapping, thus:

- L_1 = {Peter likes Mary, Jim likes_g the dog and Paul likes_g the cat} and,
- $L_2 = \{\epsilon\}$

then $L_1[\text{likes\_g} := L_2]_{IO}$ = {Peter likes Mary, Jim the dog, and Paul the cat}.

References

[BS11] Bourreau, P., Salvati, S.: A Datalog recognizer for almost affine λ-CFGs. In: Kanazawa, M., Kornai, A., Kracht, M., Seki, H. (eds.) MOL 12. LNCS, vol. 6878, pp. 21–38. Springer, Heidelberg (2011)

[dG01] de Groote, P.: Towards abstract categorial grammars. In: Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, pp. 148–155 (2001)

[dGP04] de Groote, P., Pogodalla, S.: On the expressive power of abstract categorial grammars: Representing context-free formalisms. Journal of Logic, Language and Information 13(4), 421–438 (2004)

[Fis68a] Fischer, M.J.: Grammars with macro-like productions. In: IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory, pp. 131–142. IEEE (1968)

[Fis68b] Fischer, M.J.: Grammars with macro-like productions. PhD thesis, Harvard University (1968)

[Jos85] Joshi, A.K.: Tree-adjoining grammars: How much context-sensitivity is required to provide reasonable strucutral descriptions? In: Natural Language Parsing: Psychological, Computational and Theoretical Perspectives, pp. 206–250 (1985)

[JSW91] Joshi, A.K., Shanker, V.K., Weir, D.J.: The converence of mildly context-sensitive grammar formalisms. In: Sells, P., Shieber, S.M., Wasow, T. (eds.) Foundational Issues in Natural Language Processing, pp. 31–81. The MIT Press (1991)

[Kal10] Kallmeyer, L.: On mildly context-sensitive non-linear rewriting. Research on Language and Computation 8(2), 341–363 (2010)

[Kan06] Kanazawa, M.: Abstract families of abstract categorial grammars. In: Proceedings of WoLLIC, Stanford University CSLI (2006)

[Kan07] Kanazawa, M.: Parsing and generation as Datalog queries. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, pp. 176–183. Association for Computational Linguistics (2007)

[Kan10] Kanazawa, M.: Second-order abstract categorial grammars as hyperedge replacement grammars. Journal of Logic, Language and Information 19(2), 137–161 (2010)

[Mus01] Muskens, R.: Lambda Grammars and the Syntax-Semantics Interface. In: van Rooy, R., Stokhof, M. (eds.) Proceedings of the Thirteenth Amsterdam Colloquium, Amsterdam, pp. 150–155 (2001)

[Sal06] Salvati, S.: Encoding second-order ACGs with deterministic tree walking transducers. In: Proceedings of Formal Grammar, Malaga, Spain (2006)

[Sal11] Salvati, S.: MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is captured by the IO and the OI hierarchies. Technical report, INRIA (2011)

[SMMK91] Seki, H., Matsamura, T., Mamoru, F., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88(2), 191–229 (1991)

[Sta96] Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)

[Ven87] Venkateswaran, H.: Properties that characterize LOGCFL. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 141–150. ACM (1987)

[VSWJ87] Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, Stanford (1987)

[Wei88] Weir, D.: Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania (1988)

[Yos06] Yoshinaka, R.: Linearization of affine abstract categorial grammars. In: Proceedings of the 11th Conference on Formal Grammar, Malaga, Spain, pp. 185–199 (2006)