Categorial grammars with iterated types form a Strict Hierarchy of k-valued Languages

Denis Béchet[1], Alexandre Dikovsky[1], and Annie Foret[2]

[1] LINA UMR CNRS 6241, Université de Nantes, France
Denis.Bechet@univ-nantes.fr,
Alexandre.Dikovsky@univ-nantes.fr
[2] IRISA, Université de Rennes 1, France
Annie.Foret@irisa.fr

Abstract. The notion of k-valued categorial grammars where a word is associated to at most k types is often used in the field of lexicalized grammars as a fruitful constraint for obtaining several properties like the existence of learning algorithms. This principle is relevant only when the classes of k-valued grammars correspond to a real hierarchy of languages. Such a property had been shown earlier for classical categorial grammars.
This paper establishes the relevance of this notion when categorial grammars are enriched with iterated types.

1 Introduction

The field of natural language processing includes lexicalized grammars such as classical categorial grammars [1], the different variants of Lambek calculus [11], lexicalized tree adjoining grammars [8], etc. In these lexicalized formalisms, a k-valued grammar associates at most k categories to each word of the lexicon. For a particular model of lexicalized grammars and their corresponding languages, this definition forms a (strict) hierarchy of classes of grammars when k increases. To this hierarchy of grammars, it corresponds a growing list of classes of languages that does not necessarily form a strict hierarchy.

In fact, in the field of lexicalized grammars, the concept of k-valued grammars is often used to define sub-classes of grammars and languages that satisfy some property when the whole class does not satisfy it. In particular, this notion is important for a lot of learnability results in Gold's model [7].

In the paper, we prove that the extension of classical categorial grammars with iterated types $*$AB form strict hierarchies of classes of languages. Since Categorial Dependency Grammars [6, 3, 5] use a very similar mechanism, the result is also extended to these classes of grammars. The results give a direct justification of the notion of k-valued grammars for such systems.

The paper is organized as follows. Section 2 gives some background knowledge on categorial grammars and on iterated types. Section 3 focuses on parsing or deduction structures (the two notions are closely related for type-logical or categorial grammars). Section 4 presents the proof that the class of k-valued

categorial grammars with iteration form a strict hierarchy. Section 5 considers some variants. Section 6 concludes.

2 Background

2.1 Categorial Grammars

In categorial grammars, when a word w_1 has a type of the form $B \setminus A$, this means that w_1 can be concatenated on its left with a word of type B, so as to produce a group of words of type A. Similarly a type of the form A / B, expresses a possible concatenation on the right with a word of type B. This concatenation principle extends to groups of words. See Example 1.

Definition 1 (Types). *The* types *Tp, or formulas, are generated from a set of* primitive types *Pr, or atomic formulas, by two binary connectives*[3] *" / " (over) and "\ " (under):* $Tp ::= Pr \mid Tp \setminus Tp \mid Tp / Tp$

Definition 2 (Rigid and k-valued categorial grammars). *A* categorial grammar *is a structure $G = (\Sigma, \lambda, S)$ where:*

- *Σ is a finite alphabet (the words in the sentences);*
- *$\lambda : \Sigma \mapsto \mathcal{P}^f(Tp)$ is a function (called a lexicon) that maps a finite set of types to each element of Σ (the possible categories of each word);*
- *$S \in Pr$ is the* main type *associated to correct sentences.*

If $X \in \lambda(a)$, we say that G associates X to a and we write $G : a \mapsto X$.
A k-valued categorial grammar *is a categorial grammar where, for every word $a \in \Sigma$, $\lambda(a)$ has at most k elements. A* rigid categorial grammar *is a 1-valued categorial grammar.*

Definition 3 (Language). *Given a type calculus, based on a derivation relation $\vdash$ on Types, a sentence $v_1 ... v_n$ belongs to the language of G, written $L(G)$, provided its words v_i can be assigned types X_i whose sequence $X_1 ... X_n$ derives S according to $\vdash$.*

2.2 $*$AB Calculus

Categorial grammars usually express optional and repeatable arguments by a recursive mechanism. Here, we present a different approach that uses an extension of atomic formulas. With $*$AB Calculus, an atomic formula can be either a primitive type $x \in Pr$ or the iteration of a primitive type written $x^*, x \in Pr$. This extension lets naturally express optional repeatable dependencies. The calculus is very similar except that an iterated primitive type can be used zero, one or several times.

Categorial Dependency Grammars [6, 3, 5] use a very similar mechanism. However, in this case, types are of order one (flat) [4] , but a complex system

[3] no product connective is used in the paper

[4] the order o is null on primitive types s.t. $o(X/Y) = o(Y \setminus X) = max(o(X), 1 + o(Y))$

of polarities produces non projective dependencies. Thus, CDG is not a conservative extension of $*$AB Calculus and the reverse does not hold either.

The iterated types originate from one of the basic principles of dependency syntax, which concerns optional repeatable dependencies (cf. [12]): all modifiers of a noun share the noun as their governor and, similarly, all circonstants of a verb share the verb as their governor. At the same time, the iterated dependencies are a challenge for grammatical inference [2]. For example, as in Example 3, a repeatable circumstancial dependency A may be determined by the type $[N\backslash S/A^*]$ attached to an intransitive verb, instead of several types : $[N\backslash S]$, $[N\backslash S/A]$, $[N\backslash S/A/A]$ ...

Definition 4 (Types). *The* types *Tp, or formulas, are generated from a set of* primitive types *Pr, or* iteration of primitive types $Pr^* = \{x^*, x \in Pr\}$ *by two binary connectives " / " (over) and " \ " (under):*

$$Tp ::= Pr \mid Pr^* \mid Tp \setminus Tp \mid Tp \,/\, Tp$$

The elimination rules are as follows :

$$\left|\begin{array}{ll} X \,/\, Y, Y \vdash X & (\mathbf{L^r}) \\ X \,/\, y^*, y \vdash X \,/\, y^* & (\mathbf{L^{r*}}) \\ X \,/\, y^* \vdash X & (\mathbf{\Omega^r}) \end{array}\right| \left|\begin{array}{ll} Y, Y \setminus X \vdash X & (\mathbf{L^l}) \\ y, y^* \setminus X \vdash y^* \setminus X & (\mathbf{L^{l*}}) \\ y^* \setminus X \vdash X & (\mathbf{\Omega^l}) \end{array}\right.$$

Remark. The AB Calculus (without iteration) derivation relation is defined by the two rules L^r and L^l. AB grammars are equivalent to Context-free grammars. In more details, to each ϵ-free Context-Free Grammar G in *Greibach Normal Form*, we can associate $cg_{AB}(G)$, whose alphabet consists in the terminals of G, whose primitive types are the non terminals of G, with the following lexicon :

$a \mapsto ((\dots(X/X_n)/X_{n-1}\dots)/X_1)$ for each rule $X \to aX_1 \dots X_{n-1}X_n$ in G ; G and $cg_{AB}(G)$ have the same language ($cg_{AB}(G)$ is of order 1). For the converse direction, to each AB grammar G, we associate $cf(G)$ with the same language, having the alphabet of G as terminals, the set $Tp(G)$ of *subformulas of types of G as non-terminals*, with rules $\{B \to A\ A\backslash B \mid A\backslash B \in Tp(G)\} \cup \{B \to B/A\ A \mid B/A \in Tp(G)\} \cup \{A \to c \mid c \mapsto A \in G\}$. These equivalences are said *weak*, because they concern string languages, not structures.

Definition 5 (Head and arguments). *Any type X can be written in the following form: $((p|A_1)|...|A_n)$ where $A|B$ stands for A/B or $B\backslash A$ and p has no binary operator ; p is the* head *of X, each subtype $((p|A_1)|...A_k)$ is a* head subtype *of X, n is the* arity *of X, and each A_i is said an* argument subtype *of X.*

Example 1. Let $\lambda(John) = \lambda(Mary) = N$, $\lambda(loves) = [N\backslash S/N]$: *John loves Mary* belongs to the language (for AB or $*$AB). See also Example 3 for iteration.

CDG. The $*$AB calculus on flat types (order 1) is the basis of Categorial Dependency grammars (CDG) used for natural language. In fact CDG involve more complex types, we only give their supplementary rule D^l that handles distant dependencies (rule D^r is similar on the right): **D$^{\mathbf{l}}$**. $\alpha^{P_1(\swarrow C)P(\nwarrow C)P_2} \vdash \alpha^{P_1 P P_2}$, if the potential $(\swarrow C)P(\nwarrow C)$ satisfies the following pairing rule **FA** (*first available*): **FA** : P has no occurrences of $\swarrow C, \nwarrow C$ (see ref [5] for full details).

The relation of CDG to automata is explained below.

2.3 Abstract automata equivalent to CDG

There is a class of simple abstract automata equivalent to CDG [10] (in Russian). Intuitively, these are automata with one stack and several completely independent counters. In fact, each polarized valency of a CDG corresponds to one independent counter.

Definition 6. *A* `real-time pushdown independent counters automaton` *($RtPiCA^{(k)}, k \geq 0$) is a system $A = (W, \Gamma, Q, q_0, k, I)$, where: W is the set of input symbols (words), Γ is the set of stack symbols containing a special symbol $\perp \in \Gamma$ (bottom), Q is a set of states, $q_0 \in Q$ is the start state, $k \geq 0$, and I is a set of instructions of the form*

$$i = (aqz \rightarrow q'\alpha v)$$

in which: $a \in W$, $q, q' \in Q$, $z \in \Gamma$, $\alpha \in \Gamma^*$ and v is an integer vector of length k (empty if $k = 0$), i.e. $v \in \mathbb{Z}^k$ if $k > 0$. k is the number of counters.

Computations of $RtPiCA^{(k)}$ are defined in terms of the following transition system over configurations. A `configuration` *is a tuple (q, w, γ, V), where $w \in W^*$ (non read part of input string), $q \in Q$ (current state), $\gamma \in \Gamma^*$ (stack contents) and $V \in \mathbb{Z}^k$ (current counters' values).*

A computation step is the following transition relation:

$$< q, s, \gamma, V > \vdash_A^i < q', s', \gamma', V' >,$$

where: 1) $s = as'$;
2) $\gamma = z\gamma''$, $\gamma' = \alpha\gamma''$ γ, γ' have non-negative components ;
3) $V' = V + v$ for the instruction $i = (aqz \rightarrow q'\alpha v) \in I$.
$\vdash_A^*$ is the reflexive-transitive closure of $\vdash_A^i$.

A string $s \in W^*$ is recognized by the automaton A if $< q_0, s, \perp, (0, \ldots, 0) > \vdash_A^* < q, \varepsilon, \varepsilon, (0, \ldots, 0) >$ for some q. $L(A)$ (the language recognized by A) is the set of all strings recognized by A.

Example 2. The language $L = \{ w_1^n w_2^n w_3^n \mid n = 0, 1, \ldots \}$ is recognized by the automaton $A = (W, \Gamma, Q, q_0, k, I)$ in which: $W = \{ w_1, w_2, w_3 \}$, $Q = \{ q_0, q_1, q_2 \}$, $\Gamma = \{ z_0, w_1, w_2, w_3 \}$, $k = 1$ and the set of instructions I is as follows:

$$\begin{array}{ll} w_1\ q_0\ \perp\ \rightarrow\ q_0\ w_1 \perp\ 1 & w_1\ q_0\ w_1\ \rightarrow\ q_0\ w_1 w_1\ 1 \\ w_2\ q_0\ w_1\ \rightarrow\ q_1\ \varepsilon\ 0 & w_2\ q_1\ w_1\ \rightarrow\ q_1\ \varepsilon\ 0 \\ w_3\ q_1\ \perp\ \rightarrow\ q_2\ \perp\ -1 & w_3\ q_2\ \perp\ \rightarrow\ q_2\ \perp\ -1 \\ w_3\ q_2\ \perp\ \rightarrow\ q_2\ \varepsilon\ -1 & \end{array}$$

The equivalence of $RtPiCA^{(k)}$ and CDG is proved in [10].

Theorem 1. *A language L is recognized by a $RtPiCA^{(k)}$ A for some k if and only if it is generated by a CDG.*

3 Deduction structures

In this section we focus on structures for the calculus $*$AB (and CDG) ; in fact, these rules are extensions of the cancellation rules of classical categorial grammars that lead to the generalization of FA-structures used here.

3.1 *FA* structures over a set $\mathcal{E}$

We give a general definition of *FA* structures over a set $\mathcal{E}$, whereas in practice $\mathcal{E}$ is either an alphabet Σ or a set of types such as Tp.

Definition 7 (*FA* structures). *Let $\mathcal{E}$ be a set, a FA* structure *over $\mathcal{E}$ is a binary tree where each leaf is labelled by an element of $\mathcal{E}$ and each internal node is labelled by $\mathbf{L^r}$ (forward application) or $\mathbf{L^l}$ (backward application):*

$$\mathcal{FA}_{\mathcal{E}} ::= \mathcal{E} \mid \mathbf{L^r}(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}}) \mid \mathbf{L^l}(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}})$$

3.2 Functor-Argument Structures with Iterated Subtypes

The **functor-argument structure** and **labelled functor-argument structure** associated to a (dependency) structure proof in $*$AB (or in CDG), are obtained as below.

Definition 8. *Let ρ be a structure proof, ending in a type t. The* **labelled functor-argument structure** *associated to ρ, denoted $lfa_{iter}(\rho)$, is defined by induction on the length of the proof ρ considering the last rule in ρ:*

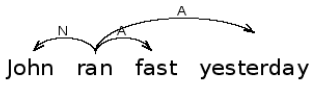
- if ρ has no rule, then it is reduced to a type t assigned to a word w, let then $lfa_{iter}(\rho) = w$;
- if the last rule is $\mathbf{L^l}$ c^{P_1} $[c \setminus \beta]^{P_2} \vdash [\beta]^{P_1 P_2}$, by induction let ρ_1 be a structure proof for c^{P_1} and $\mathcal{T}_1 = lfa_{iter}(\rho_1)$; and let ρ_2 be a structure proof for $[c \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = lfa_{iter}(\rho_2)$: then $lfa_{iter}(\rho)$ is the tree with root labelled by $\mathbf{L^l}_{[c]}$ and subtrees $\mathcal{T}_1$, $\mathcal{T}_2$;
- if the last rule is $\mathbf{\Omega^l}*$ $[c^* \setminus \beta]^{P_2} \vdash [\beta]^{P_2}$, by induction let ρ_2 be a structure proof for $[c^* \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = lfa_{iter}(\rho_2)$: then $lfa_{iter}(\rho)$ is $\mathcal{T}_2$;
- if the last rule is $\mathbf{L^{l*}}$ c^{P_1} $[c^* \setminus \beta]^{P_2} \vdash [c^* \setminus \beta]^{P_1 P_2}$, by induction let ρ_1 be a structure proof for c^{P_1} and $\mathcal{T}_1 = lfa_{iter}(\rho_1)$ and let ρ_2 be a structure proof for $[c^* \setminus \beta]^{P_2}$ and $\mathcal{T}_2 = lfa_{iter}(\rho_2)$: $lfa_{iter}(\rho)$ is the tree with root labelled by $\mathbf{L^l}_{[c]}$ and subtrees $\mathcal{T}_1$, $\mathcal{T}_2$;
- we define similarly the function lfa_{iter} when the last rule is on the right, using $/$ and $\mathbf{L^r}$ instead of $\setminus$ and $\mathbf{L^l}$;
- (in the CDG case) if the last rule is $\mathbf{D^l}$, then $lfa_{iter}(\rho)$ is taken as the image of the proof above.

The functor-argument structure $fa_{iter}(\rho)$ is obtained from $lfa_{iter}(\rho)$ (the labelled one) by erasing the labels $[c]$.

Example 3. Let $\lambda(John)=N$, $\lambda(ran)=[N\setminus S/A^*]$, $\lambda(yesterday)=\lambda(fast)=A$, then $s'_3 = \mathbf{L^l}_{[N]}(John, \mathbf{L^r}_{[A]}(\mathbf{L^r}_{[A]}(ran, fast), yesterday)$ (labelled structure) and $s_3 = \mathbf{L^l}(John, \mathbf{L^r}(\mathbf{L^r}(ran, fast), yesterday)$ are associated to ρ_1 below :

ρ_1 :

$$\dfrac{N \qquad \dfrac{\dfrac{\dfrac{[N\setminus S/A^*]\ A}{[N\setminus S/A^*]}\mathbf{I^r} \qquad A}{[N\setminus S/A^*]}\mathbf{I^r}}{[N\setminus S]}\mathbf{\Omega^r}*}{S}\mathbf{L^l}$$



(dependency structure)

4 A strict hierarchy

For each $k \in \mathbb{N}$, we are interested in classes of the form $\mathcal{C}^k_{<constraint>}$ of languages corresponding to k-valued grammars with some $<$ *constraint* $>$. This section proves for some $<$ *constraint* $>$ that such families forms a strict hierarchy (if the lexicon has at least 2 elements):

For instance, a first very easy result when we consider the $*$AB calculus (denoted by * as class constraint) is given by the fact that $\mathcal{C}^0_* \subsetneq \mathcal{C}^1_*$ because $\mathcal{C}^0_* = \emptyset$ and $\mathcal{C}^1_*$ contains the (finite) language $\{a\} = \mathcal{L}_*(G)$ for the rigid grammar $G : a \mapsto S$.

Note that the class of languages corresponding to rigid AB-grammars is a proper subset of the languages of rigid $*$AB-grammars: consider$L = \{a^+\}$ generated by $G = \{a \mapsto S \,/\, S^*\}$, which cannot be generated by a rigid AB-grammar.

4.1 Overview

We first sum up some previous work for classical categorial grammars (AB) and non-associative Lambek grammars (NL).

AB. A similar problem was solved by Kanazawa in [9] for the classes of k-valued classical categorial grammars. The proof scheme was as follows:

- Languages: for $k > 0$, $L_{AB,k} =_{def} \{a^i b a^i b a^i \mid 1 \leq i \leq 2k\}$
- Grammars:[5] for $k > 0$,

$$G_k = \begin{cases} a \mapsto x, \\ \qquad (\cdots(S \underbrace{/\,x)\cdots/\,x)}_{i}\,/\,y)\underbrace{/\,x)\cdots/\,x)}_{i}\,/\,y)\underbrace{/\,x)\cdots/\,x)}_{i-1} & (1 \leq i \leq k) \\ b \mapsto y, \\ \qquad \underbrace{(x\setminus(\cdots\setminus(x\setminus}_{i}(\cdots(S\underbrace{/\,x)\cdots/\,x)}_{i}\,/\,y)\underbrace{/\,x)\cdots/\,x)}_{i}\cdots) & (k{+}1 \leq i \leq 2k) \end{cases}$$

- The language (for AB) of G_k is $L_{AB,k}$.

[5] In fact, the second type of a can be abbreviated as $S \,/\, x^i y x^i y^{i-1}$ and the second type of b can be abbreviated as $x^i \setminus (S \,/\, x^i y x^i)$

- Property: for $k > 0$, $L_{AB,k}$ is a $(k+1)$-valued language but is not a k-valued language for classical categorial grammars.

NL. For Lambek non-associative calculus the proof scheme [4] is based on the previous one (for AB), but using grammars beyond order 1, $2k+1$ words and generalized AB-deductions. The proof scheme is as follows:

- Languages: for $k > 0$, $L_{NL,k} =_{def} \{abb\} \cup \{a^i ba^i ba^i \mid 1 \leq i \leq 2k\}$
- Grammars: $k+1$-valued grammar $G'_k = \sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S \,/\, y) \,/\, y$.
- The language (for NL) of G'_k is $L_{NL,k}$.
- Property: for $k > 0$, $L_{NL,k}$ is a $(k+1)$-valued language but is not a k-valued language for NL.

Towards Iteration. We can easily show that the languages of grammars G_k is the same when we consider the $*$AB calculus instead of the AB rules (because G_k has not iteration). The same remark holds for grammar G'_k.

This shows that the languages $L_{AB,k}$ are also $(k+1)$-valued languages for the $*$AB calculus. It is thus natural to ask whether they are k-valued for the $*$AB calculus as well. This is the purpose of next section.

Remark. One key point in the adaptation is that, when the language is finite ($L_{AB,k}$ is finite), an iterated argument subtype cannot be used in a proof tree for application of L^{l*} or L^{r*}.

4.2 Order 1 and Iteration

For each $k \in \mathbb{N}$, we can consider the class $\mathcal{C}^k_{*,flat}$ of languages corresponding to k-valued $*$-AB grammars with types of order at most 1. This section proves that this family forms a strict hierarchy (if the lexicon has at least 2 elements):

Theorem 2. $\quad \forall k \in \mathbb{N} \quad \mathcal{C}^k_{*,flat} \subsetneq \mathcal{C}^{k+1}_{*,flat}$

Before the details of proof, we introduce some definitions and remarks.

In this section, we consider the *binary deduction trees* obtained by omitting the Ω unary steps and where each node is decorated with the type that is obtained by application of the elimination rule on the immediate subtrees. These trees also correspond to the functor-argument structures previously described.

Definition 9. *We say that B is an *-context of A, when we can write:*
$B = (G^*_{i,p'_i} \backslash ... G^*_{i,1} \backslash A / D^*_{i,1} ... / D^*_{i,p_i})$ *where the sequences of iterated types (on the left, or on the right of A) are possibly empty.*

When B is an *-context of A: if $\Delta, A, \Gamma \vdash X$ then $\Delta, B, \Gamma \vdash X$ as well (using $\mathbf{\Omega^r}*$ and $\mathbf{\Omega^l}*$).

Rule patterns. We observe that each type occurring in a binary deduction tree obtained by omitting the Ω unary steps is a head subtype of some type associated to a leaf. The patterns are as follows :

(on the right - *similarly on the left* -)

$$\frac{(G^*_{i,p'_i}\backslash ...G^*_{i,1}\backslash A_i/C_i/D^*_{i,1}.../D^*_{i,p_i})\ \ C'_i}{A_i}\ L^r(\textit{several } \Omega^r \textit{ and } \Omega^l)$$

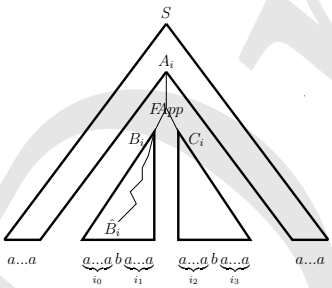
$$\frac{(G^*_{i,p'_i}\backslash ...G^*_{i,1}\backslash A_i/C^*_i/D^*_{i,1}.../D^*_{i,p_i})\ \ C'_i}{A_i}\ L^{r*}(\textit{several } \Omega^r \textit{ and } \Omega^l)$$

$$\frac{(G^*_{i,p'_i}\backslash ...G^*_{i,1}\backslash A_i/C^*_i/D^*_{i,1}.../D^*_{i,p_i})\ \ C'_i}{A_i/C^*_i}\ L^{r*}(\textit{several } \Omega^r \textit{ and } \Omega^l)$$

where C'_i is a $*$-context of C_i.

Steps of proof

1. Obviously, we have $\forall k \in \mathbb{N}\ \ \mathcal{C}^k_* \subseteq \mathcal{C}^{k+1}_*$
2. For $k > 0$, we consider $L_{*,k} =_{def} \{a^iba^iba^i \mid 1 \leq i \leq 2k\}$
3. We see that $L_{*,k}$ is a $(k+1)$-valued language : because G_k is $(k+1)$-valued, without $*$ in its types, its language is as in the AB case, which is $\{a^iba^iba^i \mid 1 \leq i \leq 2k\}$ as shown in [9].
4. We prove that $L_{*,k}$ is not a k-valued language for $*$AB languages.
   Proof : suppose G is a k-valued grammar with $*$AB language $L_{*,k}$
   (a) For each element of $L_{*,k}$, there exists a *binary deduction tree* : $\mathcal{T}_i$ for $a^iba^iba^i$ $(1 \leq i \leq 2k)$
   (b) For $0<i\leq 2k$ let A_i denote the root type of **the smallest subtree** in $\mathcal{T}_i$ **whose yield includes both** b. This gives two subtrees with one b with yields $a^{i_0}ba^{i_1}$ and $a^{i_2}ba^{i_3}$ $(i_1+i_2=i)$. Then, we consider the antecedents of A_i in $\mathcal{T}_i : C'_i$ and B_i such that : $B_i = (G^*_{i,p'_i}\backslash ...G^*_{i,1}\backslash A_i/C^\delta_i/D^*_{i,1}.../D^*_{i,p_i})$ (or $B_i = (D^*_{i,p_i}\backslash ...\backslash D^*_{i,1}\backslash C^\delta_i\backslash A_i)/G^*_{i,1}.../G^*_{i,p'_i})$) where δ is either $*$ or empty, and such that C'_i is a $*$context of C_i.



In fact, δ cannot denote $*$, otherwise, we would get deductions involving iterations of C_i (replacing one C_i) for words with more than two b. Each B_i is thus an *-context of A_i/C_i or of $C_i\backslash A_i$.
We define $\widehat{B_i}$ as the type in G "providing" B_i (following functors) in $\mathcal{T}_i$.
We define $\widehat{C'_i}$ as the type in G "providing" C'_i (following functors) in $\mathcal{T}_i$.

   (c) We remark that $\forall i : B_i \neq A_i$ *and* $C'_i \neq A_i$.
   Otherwise, if $B_i = A_i$ by replacing the subtree ending in B_i *(or* C'_i *if* $C'_i = A_i$*)* by the subtree ending in A_i, we would get a derivation of a word with three b instead of two.
   (d) More generally : $\forall i, j : A_j$ cannot have B_i or C_i as head subtype.
   Otherwise, a subtree ending in B_i (or a $*$context of C_i) would contain the subtree ending with A_j that has two b.
   (e) We prove that: $\forall i \neq j : B_i \neq B_j$
   Let $y^i_{ce}(X_i)$ denote the center part of the yield with root X_i in $\mathcal{T}_i$. (this is i_1 for the left subtree with yield $a^{i_0}ba^{i_1}$ and i_2 for the right subtree

with yield $a^{i_2}ba^{i_3}$), we have $\forall i : y^i_{ce}(B_i) + y^i_{ce}(C'_i) = i$.
- Suppose (from the contrary) (i) $B_i = B_j$, for some $i \neq j$;
Since $i \neq j$, either $y^i_{ce}(B_i) \neq y^j_{ce}(B_j)$ or $y^i_{ce}(C'_i) \neq y^j_{ce}(C'_j)$.
- - Suppose first (ii) $y^i_{ce}(B_i) \neq y^j_{ce}(B_j)$; from (ii) replacing in T_j, $(j \neq 0)$, B_j by B_i is a derivation of a word $w = ...ba^{j'}ba^j$ or $w = a^jba^{j'}b...$, where $j' = y^i_{ce}(B_i) + y^j_{ce}(C'_j)$ this word w is not in $L_{*,k}$ since $j' = y^i_{ce}(B_i) + y^j_{ce}(C'_j) \neq y^j_{ce}(B_j) + y^j_{ce}(C'_j) = j$; this contradicts the assumption that G has $L_{*,k}$ as language (for $*AB$).
- - Suppose instead (ii)' $y^i_{ce}(C'_i) \neq y^j_{ce}(C'_j)$;
- - - if (iii) $C_i = C_j$: replacing in T_j, C'_j by C'_i yields a similar word w not in $L_{*,k}$ with $j' = y^j_{ce}(B_j) + y^i_{ce}(C'_i)$ occurrences of a between the b and $j' \neq j$, (ii)' also leads to a contradiction.
- - - otherwise (iii) $C_i = D_{i,k}$ for some $D^*_{i,k}$ of B_i
$B_i = (G^*_{i,p'_i} \backslash ...G^*_{i,1} \backslash A_i / C_i / D^*_{i,1} .../ D^*_{i,p_i})$ (in the right case) ;
however in such a case, we could replace C'_i by a succession of C'_i, using the iteration rule, producing a word with more than two b.
Therefore (i) is not possible : this means that all B_i are distinct.

(f) We prove that: $\forall i, j : \widehat{B_i} \neq \widehat{B_j}$.
We write $X|Y$ as an abbreviation for $X \,/\, Y$ or for $Y \setminus X$ (functor first).
- Suppose $\widehat{B_i} = \widehat{B_j}$. One (say B_i) is a head subtype of the other (B_j), that is in the form: $B_j = ...(B_i|D'_1...)|D'_n$
with $B_j = (G^*_{j,p'_j} \backslash ...G^*_{j,1} \backslash (A_j / C_j) / D^*_{j,1} .../ D^*_{j,p_j})$ (in the right case) ;
- - if B_i is a strict [6] head subtype of A_j / C_j, we then get A_j in a subtree ending in B_i , which is impossible since the yield would then have three b instead of two.
- - otherwise, B_i is a *context[7] of A_j / C_j (in the right case), which entails that $C_i = C_j$; then, replacing B_j by B_i in $\mathcal{T}_j$ or C'_i by C'_j in $\mathcal{T}_i$ gives deduction trees: which leads to a contradiction using a reasoning similar to that of $B_i \neq B_j$. [8]

(g) As a consequence, we get a contradiction as follows.
Let $f(i)$ denote the index s.t. $\widehat{C'_i} = \widehat{B_{f(i)}}$. By definition C_i is a head subtype of $\widehat{C'_i}$ and $B_{f(i)}$ is a head subtype of $\widehat{B_{f(i)}}$, that is the same type. Therefore, one of C_i and $B_{f(i)}$ is a head subtype of the other ; because C_i is primitive and $B_{f(i)}$ is not, C_i is a head subtype of $B_{f(i)}$. This entails that C_i is a head subtype of $A_{f(i)}$ as well, which is impossible as shown previously.

5. Thus $\forall k > 0 \quad \mathcal{C}^k_{*,flat} \neq \mathcal{C}^{k+1}_{*,flat}$ (we have also seen in the introduction to the section that the property is also true for $k = 0$).

[6] (not equal to)
[7] possibly equal to
[8] $B_i = (G^*_{j,p'_{j,q'}} \backslash ...G^*_{j,1} \backslash A_j / C_j / D^*_{j,1} .../ D^*_{j,q})$ for some $q' \leq p'_j$ and $q \leq p_j$

4.3 Order >1 and iteration

The previous reasoning can be adapted to the *AB calculus where types are not necessarily flat (order >1), using the same deduction rules and structures.

Theorem 3. $\forall k \in \mathbb{N} \quad \mathcal{C}_*^k \subsetneq \mathcal{C}_*^{k+1}$

Sketch of proof. To this end, we use in this section the languages $L_{NL,k} = \{abb\} \cup \{a^iba^iba^i \mid 1 \leq i \leq 2k\}$ and consider $2k+1$ proof trees instead of $2k$ in the previous section.

- Languages: for $k > 0$, $L_{NL,k} =_{def} \{abb\} \cup \{a^iba^iba^i \mid 1 \leq i \leq 2k\}$
- Grammars: $k+1$-valued grammar $G'_k = \sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S \,/\, y) \,/\, y.$ and we can show $\mathcal{L}_*(\sigma(G_k)) = L_{NL,k}$.
- Property: for $k > 0$, $L_{NL,k}$ is a $(k+1)$-valued language (using G'_k) but is not a k-valued language (see details below) for the *AB calculus.

Details of proof. To prove that $L_{NL,k}$ is not a k valued language, we proceed as in the previous section: we suppose the existence of a k-valued gammar G', with language $L_{NL,k}$ and we consider a deduction tree $\mathcal{T}_i$ for $a^iba^iba^i$ $(1 \leq i \leq 2k)$ and $\mathcal{T}_0$ for abb. For $0 \leq i \leq 2k$, we define A_i as the root type of the smallest subtree in $\mathcal{T}_i$ with a yield including both b.

- We prove that: $\forall i \neq j : B_i \neq B_j$ (similarly to the previous subsection)
- $\forall i \neq j : \widehat{B_i} \neq \widehat{B_j}$ (details are similar to the previous subsection)
- As a consequence, we need $2k+1$ distinct $\widehat{B_i}$.
- Contradiction: $2k+1$ distinct $\widehat{B_i}$ are needed with a k-valued grammar with a useful lexicon of 2 words (a and b).

The advantage of this construction is to handle directly $2k+1$ types ($2k$ in the previous one). However, a main difference is the presence of types of order 2 in the grammar.

5 Conclusion

$*$**AB**. The paper studies variants of grammatical systems with iterated types: involving flat type (order 1) or not. We have proved that the classes of k-valued categorial grammars form a strict hierarchy of classes of languages. Thus, the notion of k-valued grammars is relevant for both systems: each $k \in \mathbb{N}$ defines a particular class of languages. The proof relies on generalized AB deductions and their corresponding functor-argument structures that enables us to define languages of structured sentences as for classical categorial grammars.

CDG. In fact, our strict hierarchy theorem also extends to categorial dependency grammars (CDG) with empty potentials, due to the following argument. A CDG-grammar G with empty potentials, has the same language, when considered as CDG-grammar or as $*$AB grammar (of order 1). Therefore the hierarchy for CDG with empty potentials cannot collapse.

Future work could concern other extensions of type logicial grammars, such as the extension of pregroups with iterated types.

References

1. Bar-Hillel, Y.: A quasi arithmetical notation for syntactic description. Language 29, 47–58 (1953)
2. Béchet, D., Dikovsky, A., Foret, A.: Two models of learning iterated dependencies. In: Proc. of the 15th Conference on Formal Grammar (FG 2010). LNCS, to appear, Copenhagen, Denmark (2010), [online] http://www.angl.hu-berlin.de/FG10/fg10_list_of_papers
3. Béchet, D., Dikovsky, A., Foret, A., Moreau, E.: On learning discontinuous dependencies from positive data. In: Proc. of the 9th Intern. Conf. "Formal Grammar 2004" (FG 2004). pp. 1–16. Nancy, France (2004)
4. Bechet, D., Foret, A.: k-valued non-associative Lambek grammars (without product) form a strict hierarchy of languages. In: Proceedings of LACL 2005, LNCS(LNAI) 3492. pp. 1–17. springer (2005)
5. Dekhtyar, M., Dikovsky, A.: Generalized categorial dependency grammars. In: Trakhtenbrot/Festschrift, pp. 230–255. LNCS 4800, Springer (2008)
6. Dikovsky, A.: Dependencies as categories. In: "Recent Advances in Dependency Grammars". COLING'04 Workshop. pp. 90–97 (2004)
7. Gold, E.: Language identification in the limit. Information and control 10, 447–474 (1967)
8. Joshi, A.K., Shabes, Y.: Tree-adjoining grammars and lexicalized grammars. In: Tree Automata and LGS. Elsevier Science, Amsterdam (1992)
9. Kanazawa, M.: Learnable Classes of Categorial Grammars. Studies in Logic, Language and Information, Center for the Study of Language and Information (CSLI) and The European association for Logic, Language and Information (FOLLI), Stanford, California (1998)
10. Karlov, B.: On properties of generalized categorial dependency grammars. In: Proc. of the 12th National Conference on Artificial Intelligence. vol. 1, pp. 283–290. Fizmatlit, Moscow, Tver, Russia (Sep 2010)
11. Lambek, J.: The mathematics of sentence structure. American mathematical monthly 65 (1958)
12. Mel'čuk, I.: Dependency Syntax. SUNY Press, Albany, NY (1988)